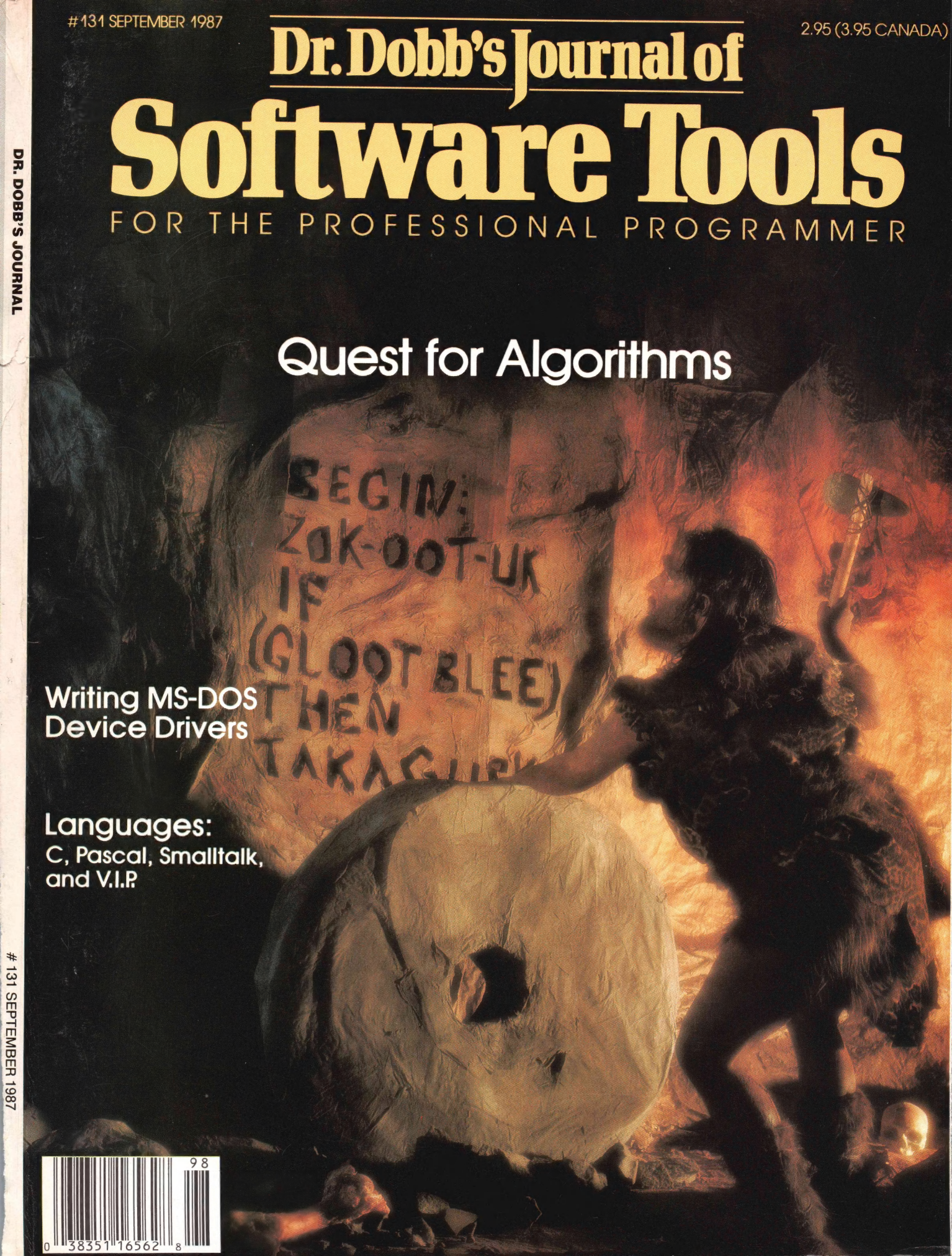
#131 SEPTEMBER 1987

2.95 (3.95 CANADA)

Dr. Dobb's Journal of Software Tools

FOR THE PROFESSIONAL PROGRAMMER

Quest for Algorithms



Writing MS-DOS Device Drivers

Languages:
C, Pascal, Smalltalk, and V.I.P.

0 38351 16562 8 98

DR. DOBB'S JOURNAL

#131 SEPTEMBER 1987

bo C: NEW! erful optimizing ler ever

Sieve benchmark

	Turbo C	Microsoft® C
Compile time	*2.4*	13.51
Compile and link time	*4.1*	18.13
Execution time	*3.95*	5.93
Object code size	*239*	249
Execution size	*5748*	7136
Price	*$99.95*	$450.00

Benchmark run on an IBM PS/2 Model 60 using Turbo C version 1.0 and the Turbo Linker version 1.0; Microsoft C version 4.0 and the MS overlay linker version 3.51.

Technical Specifications

- ☑ Compiler: One-pass optimizing compiler generating linkable object modules. Included is Borland's high-performance Turbo Linker.™ The object module is compatible with the PC-DOS linker. Supports tiny, small, compact, medium, large, and huge memory model libraries. Can mix models with near and far pointers. Includes floating point emulator (utilizes 8087/80287 if installed).
- ☑ Interactive Editor: The system includes a powerful, interactive full-screen text editor. If the compiler detects an error, the editor automatically positions the cursor appropriately in the source code.
- ☑ Development Environment: A powerful "Make" is included so that managing Turbo C program development is highly efficient. Also includes pull-down menus and windows.
- ☑ Links with relocatable object modules created using Borland's Turbo Prolog into a single program.
- ☑ Inline assembly code.
- ☑ Loop optimizations.
- ☑ Register variables.
- ☑ ANSI C compatible.
- ☑ Start-up routine source code included.
- ☑ Both command line and integrated environment versions included.
- ☑ License to the source code for Run-time Library available.

Join more than 100,000 Turbo C enthusiasts. Get your copy of Turbo C today!

Minimum system requirements: All products run on IBM PC, XT, AT, PS/2, portable and true compatibles. PC-DOS (MS-DOS) 2.0 or later. 384K RAM minimum. Basic Telecom and Editor Toolboxes require 640K.

Borland International
4585 Scotts Valley Drive, Scotts Valley, CA 95066
Telephone: (408) 438-8400 Telex: 172373

Why more than 600,000 programmers worldwide are using Turbo Pascal today

The irresistible force behind Turbo Pascal's worldwide success is Borland's advanced technology. We created a compiler so fast, that Turbo Pascal® is now the worldwide standard. And there are more tools for Turbo Pascal than for any other development environment in the world.

You'll get everything you need from Turbo Pascal and its 5 Toolboxes

Turbo Pascal and Family are all you'll ever need to perfect programming in Pascal.

If you've never programmed in Pascal, you'll probably want to start with Turbo Pascal Tutor® 2.0, and as your expertise quickly grows, add Toolboxes like our

- Database Toolbox®
- Editor Toolbox®
- Graphix Toolbox®
- GameWorks®

and our newest,

- Numerical Methods Toolbox™

And because Turbo Pascal is the established worldwide standard, 3rd party, independent non-Borland developers also offer an incredible array of programs for Turbo Pascal. ***Only $99.95!***

"Borland International's Turbo Pascal took the programming world by storm. A great compiler combined with a good editor at an astounding price, the package quickly came to be called, simply, Turbo—and has sold more than 500,000 copies.

Stephen Randy Davis, PC Magazine

Language deal of the century. *PC Magazine*"

For Scientists and Engineers: Turbo Pascal Numerical Methods Toolbox

The Numerical Methods Toolbox is a complete collection of Turbo Pascal routines and programs. Add it to your development system and you have the most comprehensive and powerful numerical analysis capabilities—at your fingertips!

The Numerical Methods Toolbox is a state-of-the-art mathematical toolbox with these ten powerful features:

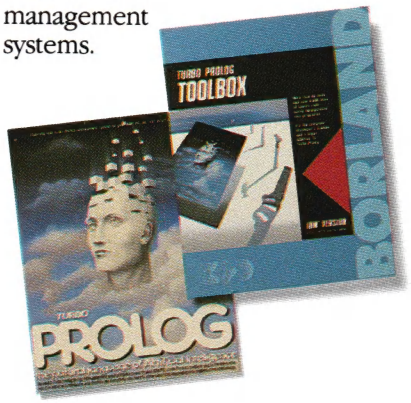
- ☑ Zeros of a function
- ☑ Interpolation
- ☑ Differentiation
- ☑ Integration
- ☑ Matrix Inversion
- ☑ Matrix Eigenvalues
- ☑ Differential Equations
- ☑ Least Squares
- ☑ Fourier Transforms
- ☑ Graphics

Each module comes with procedures that can be easily adapted to your own program. The Toolbox also comes complete with source code. So you have total control of your application.

Only $99.95!

Turbo Prolog: The Natural Language of Artificial Intelligence

Whether you're a first-time programmer or an experienced one, Turbo Prolog's natural implementation of Artificial Intelligence soon shows you how to build expert systems, natural language interfaces, customized knowledge bases and smart information management systems.



Turbo Prolog and Turbo C work hand-in-hand

Turbo Prolog® interfaces perfectly with Turbo C® because they're both designed to work with each other.

The Turbo Prolog/Turbo C combination means that you can now build powerful commercial applications using two of the most powerful languages available.

Turbo Prolog's development system includes:

- ☑ A complete Prolog compiler that is a variation of the Clocksin and Mellish Edinburgh standard Prolog.
- ☑ A full-screen interactive editor.
- ☑ Support for both graphic and text windows.
- ☑ All the tools that let you build your own expert systems and **AI** applications with unprecedented ease.

BI-1131

> "An affordable, fast, and easy-to-use language that will delight the newcomer . . . You experienced Prolog hackers will likewise be delighted, if not astonished, by the features and performance of the Turbo Prolog development environment.
>
> Turbo Prolog offers generally the fastest and most approachable implementation of that language."
>
> *Darryl Rubin, AI Expert*

How Turbo Prolog's new Toolbox adds 80 powerful tools and 8000 lines of source code

In keeping with Borland tradition, we've quickly added the new Turbo Prolog Toolbox™ to Turbo Prolog.

With 80 tools and 8000 lines of source code that can easily be incorporated into your own programs—and 40 sample programs that show you how to put these AI tools to work—the Turbo Prolog Toolbox is a highly intelligent, high-performance addition.
Only $99.95!

Turbo Prolog Toolbox features include:

- ☑ Business graphics generation: boxes, circles, ellipses, bar charts, pie charts, scaled graphics
- ☑ Complete communications package: supports XMODEM protocol
- ☑ File transfers from Reflex,® dBASE III,® 1-2-3,® Symphony®
- ☑ A unique parser generator: construct your own compiler or query language
- ☑ Sophisticated user-interface design tools
- ☑ Contains 40 example programs
- ☑ Easy-to-use screen editor: design your screen layout and I/O
- ☑ Calculated fields definition
- ☑ Over 8,000 lines of source code you can incorporate into your own programs

Tur
The most pow
compi

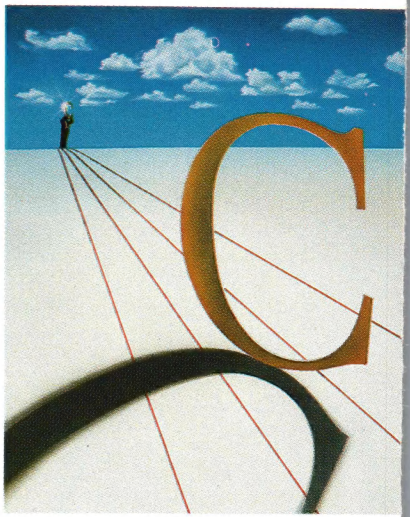
Our new Turbo C generates fast, tight, production-quality code at compilation speeds of more than 13,000 lines a minute!

It's the full-featured optimizing compiler everyone has been waiting for.

Switching to Turbo C, or starting with Turbo C, you win both ways

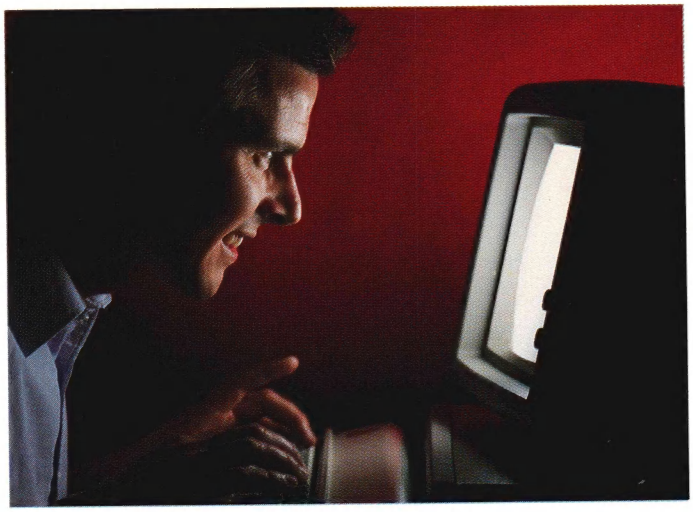
If you're already programming in C, switching to Turbo C will make you feel like you're riding a rocket instead of pedaling a bike.

If you're never programmed in C, starting with Turbo C gives you an instant edge. It's easy to learn, easy to use, and the most efficient C compiler at any price.
Only $99.95!



> "Turbo C does look like What We've All Been Waiting For: a full-featured compiler that produces excellent code in an unbelievable hurry . . . moves into a class all its own among full-featured C compilers . . . Turbo C is indeed for the serious developer . . . One heck of a buy—at any price."
>
> *Michael Abrash, Programmer's Journal*

Turbo C, Turbo Basic, Turbo Pascal and Turbo Prolog: technical excellence



"Borland International's Turbo Pascal, Turbo Basic and Turbo Prolog automatically identify themselves, by virtue of their 'Turbo' forenames, as superior language products with a common programming environment. The appellation also means to many PC users a 'must have' language. To us Turbo C looks like a coup for Borland."

Garry Ray, PC Week"

More Magic from Blaise.

Turbo C TOOLS™

Magic is easy with Turbo C TOOLS in your bag of tricks. New Turbo C TOOLS™ from Blaise Computing is a library of compiled C functions that allows you full control over the computer, the video environment, and the file system, and gives you the jump on building programs with Borland's new C compiler. Now you can concentrate on the creative parts of your programs.

The library comes with well-documented source code so that you can study, emulate, or adapt it to your specific needs. Blaise Computing's attention to detail, like the use of function prototyping, cleanly organized header files, and a comprehensive, fully-indexed manual, makes Turbo C TOOLS the choice for experienced software developers as well as newcomers to C.

Turbo C TOOLS provides the sophisticated, bullet-proof capabilities needed in today's programming environment, including removable windows, "sidekickable" applications, and general interrupt service routines written in C.

The functions contained in Turbo C TOOLS are carefully crafted to supplement Turbo C, exploiting its strengths without duplicating its library functions. As a result you'll get functions written predominantly in C, that isolate hardware independence, and are small and easy to use.

Turbo C TOOLS embodies the full spectrum of general purpose utility functions that are critical to today's applications. Some of the features in Turbo C TOOLS are:

- **WINDOWS** that are stackable and removable, that have optional borders and a cursor memory, and that can accept user input.
- **INTERRUPT SERVICE ROUTINE** support for truly flexible, robust and polite applications. We show you how to capture DOS critical errors and keystrokes.
- **INTERVENTION CODE** lets you develop memory resident applications that can take full advantage of DOS capabilities. With simple function calls, you can schedule a Turbo C function to execute either when a "hot key" is pressed or at a specified time.
- **RESIDENT SOFTWARE SUPPORT** lets you create, detect, and remove resident utilities that you write with Turbo C TOOLS.
- **FAST DIRECT VIDEO ACCESS** for efficiency, and support for all monitors including EGA 43-line mode.
- **DIRECTORY AND FILE HANDLING** support let you take advantage of the DOS file structure, including volume labels and directory structure.

In addition to Turbo C TOOLS, Blaise Computing Inc. has a full line of support products for Microsoft, Lattice and Datalight C, Microsoft Pascal and Turbo Pascal. Call today for details, and make magic!

Turbo C TOOLS supports the Borland Turbo C compiler, requires DOS 2.00 or later and is just $129.00

THE BLAISE MENU

Turbo POWER TOOLS PLUS *$99.95*
Screen and window management including EGA support; DOS memory control; ISRs; scheduled intervention code; and much more. For Turbo Pascal.

Turbo POWER SCREEN
COMING SOON! General screen management; paint screens; block mode data entry or field-by-field control with instant screen access. For Turbo Pascal.

Turbo ASYNCH PLUS *$99.95*
Interrupt driven support for the COM ports. I/O buffers up to 64K; XON/XOFF; up to 9600 baud; modem and XMODEM control. For Turbo Pascal.

PASCAL TOOLS/TOOLS 2 *$175.00*
Expanded string and screen handling; graphics routines; memory management; general program control; DOS file support and more. For MS-Pascal.

C TOOLS PLUS *$175.00*
Windows; ISRs; screen handling; multiple monitors; EGA 43-line text mode; direct screen access; DOS file handling and more. For MS and Lattice C version 3.00 and later.

LIGHT TOOLS *$99.95*
Windows; ISRs; EGA 43-line text mode; direct screen access; DOS file handling and more. For the Datalight C compiler.

ASYNCH MANAGER *$175.00*
Full featured interrupt driven support for the COM ports. I/O buffers up to 64K; XON/XOFF; up to 9600 baud; modem control and XMODEM. For C or MS-Pascal.

VIEW MANAGER *$275.00*
General screen control; paint screens; block mode data entry or field-by-field control with instant screen access. For C or MS-Pascal.

RUNOFF *$49.95*
Text formatter for all programmers; flexible printer control; user-defined variables; index generation; general macro facility. Crafted in Turbo Pascal.

EXEC *$95.00*
NEW VERSION! Program chaining executive. Chain one program from another in different languages; specify common data areas; less than 2K of overhead.

TO ORDER CALL TOLL FREE
800-333-8087
TELEX NUMBER-338139

YES! I want to make magic!
Enclosed is $_____ for _____ copies of _____.
☐ Please send me more information on your products.
CA residents add Sales Tax. Domestic orders add $4.00 for UPS shipping, $10.00 for Federal Express standard air.
Name: _____ Phone: (_____) _____-_____
Address: _____ State: _____ Zip: _____
City: _____ Exp. Date: _____
VISA or MC#: _____

Turbo C is a trademark of Borland International.

BLAISE COMPUTING INC.
2560 Ninth Street, Suite 316 Berkeley, CA 94710 (415) 540-5441

CIRCLE 159 ON READER SERVICE CARD

Turbo Basic introduces its powerful new Telecom, Editor and Database Toolboxes

NEW!

Turbo Basic® is the breakthrough you've been waiting for. The same power we brought to Pascal with Turbo Pascal has now been applied to BASIC with Turbo Basic.

Compatible with BASICA, Turbo Basic is the high-performance, high-speed BASIC you'd expect from Borland.

> "Borland has created the most powerful version of BASIC ever."
>
> *Ethan Winer, PC Magazine*

Basically, Turbo Basic is all you need

It's a complete development environment which includes an incredibly fast compiler, an interactive editor and a trace debugging system. It outperforms all its rivals, and because it's compatible with BASICA, you probably already know how to use it.

Includes a free MicroCalc™ spreadsheet complete with source code. ***Only $99.95!***

A technical look at Turbo Basic

- ☑ Full recursion supported
- ☑ Standard IEEE floating-point format
- ☑ Floating-point support, with full 8087 (math co-processor) integration. Software emulation if no 8087 present
- ☑ Program size limited only by available memory (no 64K limitation)
- ☑ VGA, CGA, and EGA support
- ☑ Access to local, static, and global variables
- ☑ Full integration of the compiler, editor, and executable program, with separate windows for editing, messages, tracing, and execution
- ☑ Compile, run-time, and I/O errors place you in the source code where error occurred
- ☑ New long integer (32-bit) data type
- ☑ Full 80-bit precision
- ☑ Pull-down menus
- ☑ Full window management

NEW!

Telecom Toolbox is a complete communications package which takes advantage of the built-in communications capabilities of BASIC—use as is or modify.

- Pull-down menus and windows
- XMODEM support
- VT 100 terminal emulation
- Captures text to disk or printer
- PhoneBook file
- 300, 1200, 2400 baud support
- Supports script files
- Fast screen I/O
- Supports most of XTalk's command set
- Manual dial and redial options

Use Telecom Toolbox to embed communications capabilities into your own programs and/or build your own communications package. Source code included for all Toolbox code and sample programs. ***Only $99.95!***

For the dealer nearest you or to order by phone call

(800) 255-8008

in CA (800) 742-1133 in Canada (800) 237-1136

BORLAND INTERNATIONAL

SUMMER BREAK SPECIAL ! Buy Turbo Basic and get a FREE product. See your dealer for details!

NEW!

Database Toolbox means that you don't have to reinvent the wheel each time you write new Turbo Basic database programs.

- ☑ "Trainer" shows you how B+ trees work. (Simply key in sample records and you'll see your index being built.)
- ☑ Turbo Access instantly locates, inserts or deletes records in a database—using B+ trees.
- ☑ Turbo Sort sorts data on single items or on multiple keys and features virtual memory management for sorting large data files.

Source code included.
Only $99.95!

NEW!

Editor Toolbox is all you need to build your own text editor or word processor. Includes source code for two sample editors.

First Editor is a complete editor ready to include in your programs, complete with windows, block commands and memory-mapped screen routines.

MicroStar™ is a full-blown text editor with a complete pull-down menu user interface, and gives you

- Wordwrap
- Undo last change
- Auto-Indent
- Find and Find/Replace with options
- Set left/right margins
- Block mark, move and copy
- Tab, insert, overstrike modes, line center etc.

Includes source code.
Only $99.95!

BI-1131A

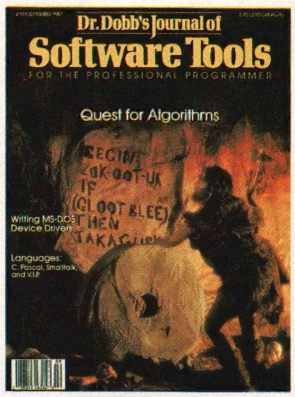
CIRCLE 161 ON READER SERVICE CARD

ARTICLES

COLUMNS

FORUM

PROGRAMMER'S SERVICES



About the Cover

It's a problem we've all faced, and it dates back to Og, the caveman. You struggle with primitive tools and hostile environments, only to discover that someone else has already invented the wheel. Special thanks to Bob Wynne, our production manager, for battling saber-toothed tigers every month to put this magazine in your hands—and for taking the time to model for the cover photo. All through the photo session, we heard him muttering to himself, "What goes round, comes round . . . ."

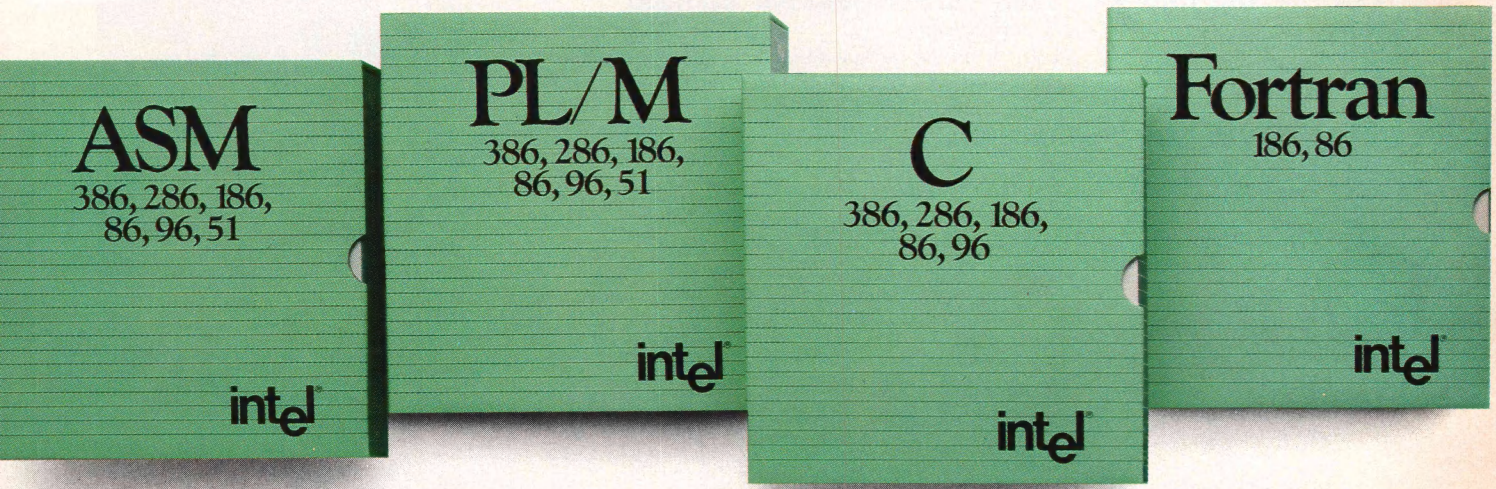
This Issue

In case you hadn't guessed, it's algorithm time again. We've got a good selection of items for your toolbox: everything from going in circles to climbing trees. Now all you need is a generic data structure (one size fits all).

Next Issue

October's *DDJ* is our annual Forth issue. Among the articles is a demonstration of how threaded subroutines let a 68000 Forth approach the speed of compiled languages. We also have a special feature on hacking the AppleTalk network to add remote nodes via an ordinary RS-232 serial link.

WRITE FASTER IN ANY LANGUAGE.



If you develop software for any product based on an Intel microcontroller or microprocessor, including the 80386, the unique debug hooks in the Intel languages will help get the job done faster.

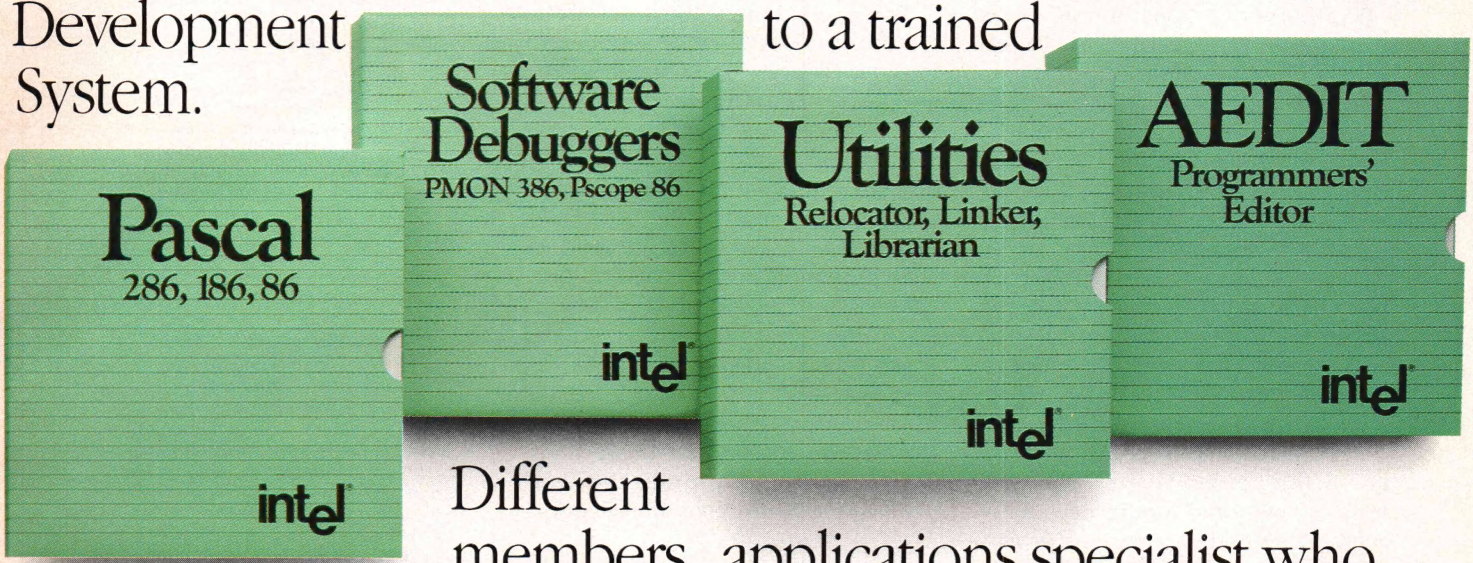
In fact, when used with Intel debuggers and emulators, Intel development languages can provide more debug data than any other high-level language.

Debug hooks let you symbolically debug in the same high-level language you wrote in without having to deal with machine or hex code. Which means 80.386 reads as 80.386, not 50 62 D0 C5.

Because the location of both code and data are easily specified with our locator, it is easier for you to develop ROM-based firmware.

Since Intel languages

produce identical object code regardless of the host, you can write code at a PC running DOS, a VAX*/VMS terminal, or an Intel Development System.



Different members of the same design team can therefore choose the most effective combination of languages and systems to get the job done faster.

Intel post-sales support can also help you get the job done faster. We invented the microprocessor. We know microprocessors and languages for Intel architectures better than anyone else.

When you buy an Intel language, you have access to our customer hotline. So if you ever have a question you can talk directly to a trained applications specialist who understands our products. And can give you the right answers. Faster.

To order today, or get more information—including a free catalog of our development tools—call toll-free 1-800-87-INTEL.

The sooner you call, the faster you'll get the job done.

intel®

*VAX is a registered trademark of Digital Equipment Corporation.

EDITORIAL

Stone Age Computing

This month's *DDJ* has, without argument, one of the most thought-provoking covers we've produced. The mix of Og, the caveman, and a pseudocoded cave wall is incongruous enough to provoke a hoard of questions in the reader's mind. What does that code on the wall really say? Why is Og staring so intently at the bottom line of the algorithm? And perhaps most important of all: has the staff at *DDJ* finally gone round the bend?

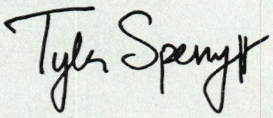
Once you get past the fun captions and lines about "recoding the wheel," however, there's an aspect of Og's situation that is jarringly similar to the present day. Simply put: today's personal computers often seem (despite their sexy new processors and sophisticated designs) to be the product of Neanderthal thinking. Again and again we're presented with products that are inexplicably primitive or crippled.

Apple's Macintosh, for example, seems to be a curious mix of high-tech and high-Bronze Age. Much of the design work in the Macintosh is wasted in a machine aimed at people only slightly more sophisticated than our friend Og. We have icons for all those computer users who are too stupid or lazy to read English and a single-button mouse for those unable to handle a keyboard. To its credit, Apple has recently added an innovation for the Mac's most advanced users—the keyboards now have cursor keys.

The problems of primitive thinking and design are hardly an Apple monopoly, of course. Microsoft's MS-DOS, based on ideas from CP/M and Unix, is so primitive that it defies tasteful jokes. On the other hand, OS/2—with over a million lines of code yet to be debugged by adventurous users—has tremendous potential for being as good a running gag as Microsoft's Windows. Assuming, of course, it actually runs.

And then there's the Intel architecture.... Has anyone out there found a solution to switching back and forth between protected and real mode in the 80286? Being able to use both modes was going to be a major feature of the chip, but here we are—years after the introduction—and the accepted method of switching modes is to reset the processor. The method used by Microsoft in OS/2 (a reset request via the keyboard controller) seems to be the best kludge so far. That is, it's painfully slow, but it works. You'd think there'd be something less, well, "Stone Age" than putting the chip to sleep for a couple of milliseconds.

In flaming, it's generally a good idea to keep your historical perspective. This editorial was partially inspired by a letter from one of our readers, John Dvorak. Always eager to stir things up, John questioned our ability to be nasty and rude. This editorial could be seen as a reply. Flaming is certainly a fun way to fill space in a magazine, but at *DDJ* we try to remember that the purpose is to shed light, not just to blow smoke. Or, as Mrs. Og said when she saw the wheel for the first time, "That's very nice, dear. But what's it *for?*"



Tyler Sperry
editor

Dr. Dobb's Journal of Software Tools
FOR THE PROFESSIONAL PROGRAMMER

Editorial
Editor-in-Chief *Michael Swaine*
Editor *Tyler Sperry*
Managing Editor *Vince Leone*
Associate Editor *Ron Copeland*
Assistant Editor *Sara Noah Ruddy*
Technical Editors *Allen Holub, Richard Relph*
Contributing Editors *Namir Shammas, Ernest R. Tello*
Copy Editor *Rhoda Simmons*
Production
Production Manager *Bob Wynne*
Art Director *Michael Hollister*
Assoc. Art Director *Joe Sikoryak*
Technical Illustrator *Frank Pollifrone*
Typesetter *Mary Lopez*
Cover Photographer *Michael Carr*
Circulation
Circulation Director *Maureen Kaminski*
Book Marketing Mgr. *Jane Sharninghouse*
Subscription Supervisor *Kathleen Shay*
Newsstand Sales *Larry Hupman*
Administration
Finance Director *Kate Wheat*
Business Manager *Betty Trickett*
Accounts Payable Supv. *Mayda Lopez-Quintana*
Accts. Receivable Supv. *Laura DiLazzaro*
Advertising Director
Ferris Ferdon (415) 366-3600
Account Managers *see page 137*
Promotions/Srvcs. Mgr. *Anna Kittleson*
Advertising Coordinator *Donna Rogers*
Associate Publisher
Michael Swaine
Assistant *Sara Noah Ruddy*

Dr. Dobb's Journal of Software Tools (USPS 307690) is published monthly by M&T Publishing Inc., 501 Galveston Dr., Redwood City, CA 94063; (415) 366-3600. Second-class postage paid at Redwood City and at additional entry points. *DDJ* is published under license from People's Computer Company, 2682 Bishop Dr., Suite 107, San Ramon, CA 94583, a nonprofit corporation.

Article Submissions: Send manuscripts and disk (with article and listings) to the Editor.

***DDJ* on CompuServe:** Type GO DDJ

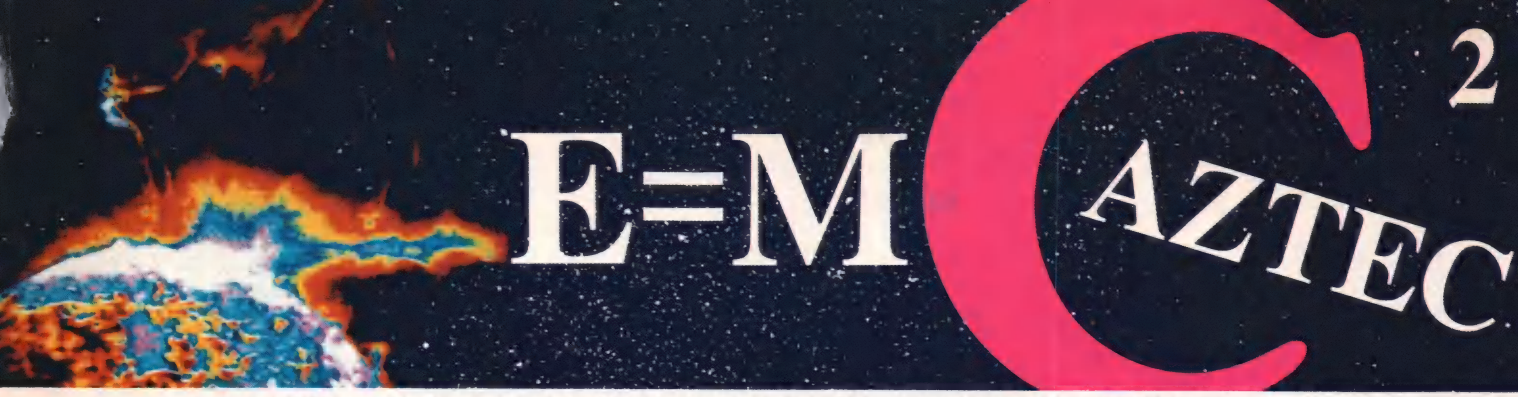
Address Correction Request: Postmaster: Send Form 3579 to *Dr. Dobb's Journal*, P.O. Box 27809, San Diego, CA 92128. **ISSN 088-3076**

Customer Service: For subscription problems call: outside CA (800) 321-3333; in CA (619) 485-9623 or 566-6947. For book/software order problems call (415) 366-3600.

Subscriptions: $29.97 per 1 year; $56.97 for 2 years. Canada and Mexico add $27 per year airmail or $10 per year surface. All other countries add $27 per year airmail. Foreign subscriptions must be prepaid in U.S. funds drawn on a U.S. bank. For foreign subscriptions, TELEX: 752-351.

Foreign Newsstand Distributor: Worldwide Media Service Inc., 386 Park Ave. South, New York, NY 10016; (212) 686-1520 TELEX 620430 (WUI).

M&T Publishing Inc.
Chairman of the Board *Otmar Weber*
Director *C.F. von Quadt*
President and Publisher *Laird Foshay*



Our thanks to NASA for supplying this computer enhanced ultraviolet photo taken by Skylab IV of a solar prominence reaching out 350,000 miles above the sun's surface

Genius Begins With A Great Idea ...

But The Idea Is Just The Beginning

What follows is the time consuming task of giving form and function to the idea.

That's why we concentrate on building into our software development systems functions and features that help you develop your software ideas in less time and with less effort.

We've started 1987 by releasing new versions of our MS-DOS, Macintosh, Amiga, ROM, and Apple // C development systems. Each system is packed with new features, impressive performance, and a little bit more genius.

FREE 2-DAY DELIVERY!

Aztec C86 4.1

New PC/MS-DOS • CP/M-86 • ROM

Superior performance, a powerful new array of features and utilities, and pricing that is unmatched make the new Aztec C86 the first choice of serious software developers.

Aztec C86-p Professional System $199
• optimized C with near, far, huge, small, and large memory + Inline assembler + Inline 8087/80287 + ANSI support + Fast Float (32 bit) + optimization options • Manx Aztec 8086/80x86 macro assembler • Aztec overlay linker (large/small model) • source level debugger • object librarian • 3.x file sharing & locking • comprehensive libraries of UNIX, DOS, Screen, Graphics, and special run time routines.

Aztec C86-d Developer System $299
• includes all of Aztec C86-p • Unix utilities make, diff, grep • vi editor • 6+ memory models • Profiler.

Aztec C86-c Commercial System. $499
• includes all of Aztec C86-d • Source for library routines • ROM Support • CP/M-86 support • One year of updates.

Aztec C86 Third Party Software

A large array of support software is available for Aztec C86. Call or write for information. The following is a list of the most requested products: • Essential Graphics • C Utility Library • Curses • Greenleaf Communication, General, and Data Window • Halo • Panel+ • PC-lint • PforCe • Pre-C • Windows for C • Windows for Data • C terp • db_Vista • db-Query • Phact • Plink-86 Plus • c-tree • r-tree • Pmate

CP/M • TRS-80 • 8080/Z80 ROM

C compiler, 8080/Z80 assembler, linker, librarian, UNIX libraries, and specialized utilities.

Aztec C II-c (CP/M-80 & ROM). $349
Aztec CII-d (CP/M-80) $199
Aztec C80 (TRS-80 3&4) $199

Aztec C68k/Am 3.4

New Amiga Release

Amiga user groups across the USA voted Aztec C68k/Am release 3.3 the best Software Development System for the Amiga. Release 3.4 is more impressive.

Aztec C68k/Am-p Professional $199
A price/feature/performance miracle. System includes: optimized C • 68000/680x0 assembler • 68881 support • overlay linker • UNIX and Amiga libraries • examples.

Aztec C68k/Am-d Developer $299
The best of Manx, Amiga, and UNIX. System includes: all of Aztec C68k/Am-p • the Unix utilities make, diff, grep and vi.

Aztec C68k/Am-c Commercial $499
Aztec C68k/Am-d plus source for the libraries and one year of updates.

FREE 2-DAY DELIVERY!

Aztec C68k/Mac 3.4

New Macintosh Release

For code quality, reliability, and solid professional features, Aztec C for the Macintosh is unbeatable. This new release includes features and functions not found in any other Macintosh C development system.

Aztec C68k/Mac-p Professional $199
• optimized C • 68000/680x0 assembler • 68881 support • overlay linker • UNIX and Macintosh libraries • examples.

Aztec C68k/Mac-d Developer. $299
The best of Manx, Macintosh, and UNIX. System includes: all of Aztec C68k/Am-p • the Unix utilities make, diff, grep • vi editor.

Aztec C68k/Mac-c Commercial $499
Aztec C68k/Am-d plus source for the libraries and one year of updates.

Aztec C65

New ProDOS Release

Aztec C65 is the only commercial quality C compiler for the Apple II. Aztec C65 includes C compiler, 6502/65C02 assembler, linker, library utility, UNIX libraries, special purpose libraries, shell development environment, and more. An impressive system.

Aztec C65-c Commercial $299
• runs under ProDOS • code for ProDOS or DOS 3.3

Aztec C65-d Developer. $199
• runs under DOS 3.3 • code for DOS 3.3

Aztec ROM Systems

6502/65C02 • 8080/Z80 • 8086/80x86 • 680x0

An IBM or Macintosh is not only a less expensive way to develop ROM code, it's better. Targets include the 6502/65C02, 8080/Z80, 8086/80x86, and 680x0.

Aztec C has an excellent reputation for producing compact high performance code. Our systems for under $1,000 outperform systems priced at over $10,000.

Initial Host Plus Target $750
Additional Targets $500
ROM Support Package $500

Vax, Sun, PDP-11 ROM HOSTS

Call for information on Vax, PDP-11, Sun and other host environments.

C' Prime

PC/MS-DOS • Macintosh
Apple II • TRS-80 • CP/M

These C development systems are unbeatable for the price. They are earlier versions of Aztec C that originally sold for as much as $500. Each system includes C compiler, assembler, linker, librarian, UNIX routines, and more. Special discounts are available for use as course material.

C' Prime $75

Aztec Cross Development Systems

Most Aztec C systems are available as cross development systems. Hosts include: PC/MS-DOS, Macintosh, CP/M, Vax, PDP-11, Sun, and others. Call for information and pricing.

How To Become An Aztec C User

To become a user call 800-221-0440. From NJ or international locations call 201-542-2121. Telex: 4995812 or FAX: 201-542-8386. C.O.D., VISA, MasterCard, American Express, wire (domestic and international), and terms are available. One and two day delivery available for all domestic and most international destinations.

Aztec C is available directly from Manx and from technically oriented computer and software stores. Aztec Systems bought directly from Manx have a 30 day satisfaction guarantee.

Most systems are upgradable by paying the difference in price plus $10. Site licenses, OEM, educational, and multiple copy discounts are available.

To order or for more information call today.

1-800-221-0440

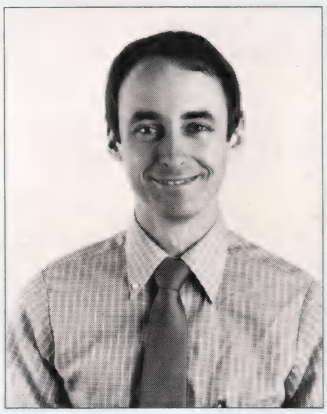
In NJ or international call (201) 542-2121 • TELEX: 4995812

MANX

CIRCLE 108 ON READER SERVICE CARD

Manx Software Systems
1 Industrial Way, Eatontown, NJ 07724

RUNNING LIGHT



I am often struck by the similarity between editing a magazine and the plight of Billy Pilgrim in *Slaughterhouse Five*. Vonnegut's hero, you may recall, had become "unstuck in time" and began to experience the moments of his life—both past and future—out of the normal sequence. A day spent in old age might be followed by a week of his youth, the shift coming without any apparent plan or warning.

Editing a magazine is a lot like that.

From the writer's perspective, things are pretty straightforward. You write and debug your program, write an article about it, and send it in to DDJ. After an awfully long time, you get a letter telling you if we liked the manuscript enough to print it.

The view from the editor's desk is a lot different. As I write this column, it's the middle of July, and this piece is literally the last thing to go into the magazine before we ship it out the door in preparation for printing. As soon as I finish this column, there are neglected Forth articles to edit for the October issue. All this happens when I should be soliciting authors and articles for the December operating systems issue or at least checking on the progress of the authors and editors working on stuff for the November graphics issue. All this high-speed flipping through the calendar is enough to drive a normal person quite mad. Having spent so much time with computers, of course, I am already certifiable.

I mention all this chaos, not in an appeal for sympathy—although it *would* be nice if they'd quit sticking pins in the voodoo doll—but so that you'll understand why it sometimes takes months to evaluate a manuscript. It's the old business of trying to drain the swamp when the alligators keep demanding your attention. Sometimes a week will pass and I'll suddenly realize I haven't looked at the latest batch of manuscripts, or checked into the DDJ forum on CompuServe. Worse, even after we've considered a piece, there's a good chance it will have to be sent off to a referee or Technical Editor.

Things are getting better, and we're speeding up the evaluation cycle. Ron Copeland has joined our staff as Associate Editor, and has already been an immense help both in tracking down manuscripts that were lost on my desk and in routing them to the proper referees. Richard Relph, who coordinated our infamous giant C Compiler roundup two years running and has written a number of articles for us, has just joined the staff as a Technical Editor. By the time you read this, our response time on manuscripts should have dropped to less than a month.

So, to those of you who got caught in the old delay loops, my apologies. For those of you who haven't sent in an article or query, there's no longer any excuse. If you've done something particularly neat or solved a particularly vexing problem, let our readers know about it by submitting an article. If you want to discuss an article idea with me, give me a call at (415) 366-3600. You can also send me e-mail via CompuServe (76703,4266), or BIX (with the clever *nom-de-baud* of "tyler").

Tyler Sperry

Tyler Sperry
editor

ARCHIVES

Intel Needs Help

"From the January-February issue of Solutions, a public-relations magazine put out by Intel Corp., we learn that Intel has prepared a Support Library for their 8087 numeric co-processor. It 'consists' of the function library, a decimal conversion module, . . . and a full 8087 software emulator for debugging without the 8087 component. The library . . . supports the proposed IEEE Floating Point Standard.

"Well, that's a lot of software (even if the 8087 does do most of the work), so we suppose that Intel's asking price of $1,250 (gulp!) is justified. But is the 8087 so hard to work with that you need twelve hundred dollars worth of interface code? Surely not! Obviously, Intel doesn't know how to sell chips. Let's show them how it's done: let's put some 8087 knowledge into the public domain so that experimenters can use the device. Many readers working with it? What kind of programming does it take?"—*D. E. Cortesi, Dr. Dobb's Clinic,* DDJ, *June 1982.*

Ten Years Ago in *DDJ*

"OLDIE BUT GOODIE AVAILABLE

"Our glorious(?) Editor spent much of a decade (no pun intended) in mutually supportive relationships with the 8-Family of Maynard, and has long had a warm spot in his heart for these quaint machines. (For the younger generation and the uninitiated, this eccentric discussion concerns one of the first—and, for many years, the most popular—minicomputers: the PDP-8 manufactured by Digital Equipment Corporation (DEC) in Maryland, MA.)

"Back in the Summer of '76, the blossoming of DDJ, the Computer Faire, the Silicon Gulch Gazette, the ACM, teaching at Stanford, NCC in Dallas, PC'77 . . . . and the list goes on and on. And our Editor has had little time to do more than turn his old friend on from time to time, and watch it glow and blink. Lest this old 8/1 feel spurned and lonely —for it is an honorable machine with a strong Puritan work ethic, not accustomed to having its diodes idle—Jim is seeking a good home for it. Price for the whole system FOB Menlo Park, California is $4,000."—*Jim Warren,* DDJ *September 1977.*

DR. DOBB'S JOURNAL of
COMPUTER
Calisthenics & Orthodontia
Running Light Without Overbyte

The Fastest C

We challenged Microsoft-C to a C compiler duel, measuring compile, link, and execution times. They wouldn't take on Optimum-C — they own it and they know how fast it is!

DATALIGHT's new Optimum-C compiler can make your programs run up to 30% faster than other compilers. That's because it's a true optimizing compiler. While many manufacturers may claim to have optimizing compilers, they can't stand up against Optimum-C. Their optimizers look at only one or maybe a few statements at a time; Optimum-C takes in the BIG picture, looking at an entire function at once.

Optimum-C tunes each function to the machine by maximizing register usage while minimizing memory usage. Optimizations performed include: global common subexpression elimination, register allocation by coloring, copy/constant propagation, loop induction variables, and dead code elimination.

Develop Programs Faster

To save compile time during code development, you can turn off the optimization. Then, when your program is developed and debugged, you can turn on the optimization and recompile for the fastest execution speed.

Also included to help you with code development is the EZ interactive editor/environment. With EZ you can compile, link, execute and quickly correct any syntax errors by letting EZ show you the erroneous source line and error message.

Library Source Code and Other Extras

Top speed is only one of the advantages you get with Optimum-C. You also get complete library/startup source code (at no additional charge) support for 4 memory models (including large model), inline 8087 code, and standard object files. Plus a complete UNIX style MAKE program and a fast screen I/O package.

Speed ROM Development

It used to be that to develop ROM code you had to purchase a compiler from one manufacturer, a debugger from another, and other tools from still another. DATALIGHT has eliminated all the hassle by providing all the support tools necessary for ROM development.

BENCHMARKS

	Sieve	Pointer	Optimize
Optimum-C	49.3	45.7	00.9
MS C	58.6	90.2	38.6
Turbo-C	62.1	49.0	40.2

ROM-*it*, the Unique ROM Development Kit

Only DATALIGHT offers you the kind of support you'll find in Rom-*it*, the new ROM support package. This unique package speeds up the delivery of your ROM application by providing many program elements that you used to have to write for yourself.

Rom-*it* includes a startup routine that can take an 8086 from restart to C program execution by setting up segment registers, stack, heap, copying initialized data from ROM to RAM, and zeroing uninitialized data. The BLAZE locator/Intel hex file generator can locate code and data anywhere in memory, while performing checks for ROM overruns, illegal data access, and complete program location. The ROMable library contains the ROMable portions of the Optimum-C library, with support for interrupt handling in C, reading and writing I/O ports, and full math support.

Debug ROMed Code from the PC

You can download, execute, and debug programs on the target system from your PC, with Remote-DSD. This debugger is a symbolic windowing debugger that shows you the C source code (on the PC) that is executing on the target system. You can view and change global variables on the target system, modify data, set breakpoints and watch registers and flags.

Try Optimum-C risk free

Call us TOLL FREE today for your copy of Optimum-C. You will also receive free* "C: A Programming Workshop" to get you started with C. To find out how we can help you get your ROM projects completed on time call us.

Optimum-C w/ C Tutorial	$139
Rom-*it*	$159

Add $7 for shipping in US/$20 outside US

COD (add $2.50)

Not Copy Protected

ORDER TOLL-FREE TODAY!

1-800-221-6630

ATTENTION OEMs!

Contact us regarding arrangements.

*Limited offer available exclusively to readers who purchase directly from Datalight.

Microsoft and MS-DOS are registered trademarks of the Microsoft Corporation. Turbo C is a registered trademark of Borland International.

Magazine Reviewers Shocked by DATALIGHT's Performance...

"Reviewing this compiler was quite a surprise for us. For such a low price, we were expecting a "lightweight" compiler. What we got was a package that is as good as or better than most of the "heavyweights." Datalight C implements a complete C language. It also compiles quickly, doesn't take up much disk space, and looks impressive in the benchmarks."

***DR. DOBBS*, August 1986**

"This is a sharp compiler!... what is impressive is that Datalight not only stole the compile time show completely, but had the fastest Fibonacci executable time and had excellent object file sizes to boot!"

***COMPUTER LANGUAGE*, February 1986**

Optimum-C Version 3.08

NEW!
EZ Interactive Development Environment
NEW!
Inline 8087/80287 Math Support

- Full UNIX System 5 C language plus ANSI extensions
- Fast/tight code via powerful optimizations including common subexpression elimination
- DLC one-step compile/link program
- Multiple memory model support
- UNIX compatible library with PC functions
- Compatible with DOS linker and assembler
- Third-party library support
- Automatic generation of .COM files
- Supports DOS pathnames, wild cards, and Input/Output redirection
- Compatible with Lattice C version 3.x
- Interrupt handling in C
- Debugger support
- ROMable code support/start-up source
- Full UNIX MAKE Program
- Tools in Source Code: cat, diff, ...

MS-DOS® Support Features

- Mouse support
- Sound support
- Fast screen I/O
- Interrupt handler

Datalight

17505-68th Avenue NE, Suite 304
Bothell, Washington 98011 USA
(206) 367-1803

CIRCLE 203 ON READER SERVICE CARD

On April 2, 1987 IBM and Quarterdeck announced the next generation in personal computing.

Introducing DESQview 2.0. Improving the past and ready for the future right now.

In one sweeping announcement from Miami Beach and New York City, IBM established new standards of performance for personal computers, with its new Personal System/2.™ Quarterdeck was there with IBM and simultaneously helped establish new standards for multi-tasking and multi-windowing.

We were there for them then. We're here for you now.

If you use two or more software programs, if you use a PC-compatible machine, if you own a new 80386 computer, if you've just bought one of the new Personal System/2 computers, or if you've tried Microsoft Windows and were disappointed but still need the power of graphics programs, DESQview 2.0 is the answer.

Consider this. InfoWorld voted DESQview's earlier version 1986 Product of the Year. SoftSector gave it the Editor's Choice Award. In PC Tech Journal's "System Builder Contest" at Comdex Fall it was voted best operating environment. And 450,000 dedicated users on four continents have voted yes with their dollars.

The new DESQview 2.0 is an order of magnitude better.

This unique software program enhances the power of your personal computer and makes it more convenient to use. It still gives your PC the power of many PCs. It still does windows. It still multi-tasks. It still breaks the DOS 640K barrier. It still transfers data. It still dials your phone. It still gives you menus for DOS. It still remembers your keystrokes (macros). It still runs your existing programs and your new programs soon to come. In fact now you can even run Windows-, GEM-, and Topview-specific programs too. And with 386 machines and our Expanded Memory Manager it still becomes a 386 control program, but now you can run text *and* CGA graphics programs in background.

The new DESQview 2.0.

For us it's the next logical step.
For you it's windows of opportunity.

SYSTEM REQUIREMENTS
• IBM Personal Computer and 100% compatibles (with 8086, 8088, 80286 or 80386 processors) with monochrome or color display; IBM Personal System/2 • Memory: 640K recommended; for DESQview itself 0-145K • Expanded Memory (Optional): expanded memory boards compatible with the Intel AboveBoard; enhanced expanded memory boards compatible with the AST RAMpage • Disk: Two diskette drives or one diskette drive and a hard disk • Graphics Card (Optional): Hercules, IBM Color/Graphics (CGA), IBM Enhanced Graphics (EGA), IBM Personal System/2 Advanced Graphics (VGA) • Mouse (Optional): Mouse Systems, Microsoft and compatibles • Modem for Auto-Dialer (Optional): Hayes or Compatible • Operating System: PC-DOS 2.0–3.3; MS-DOS 2.0–3.2 • Software: Most PC-DOS and MS-DOS application programs; programs specific to TopView 1.1, GEM 1.1 and Microsoft Windows 1.03 • Media: DESQview 2.0 is available on either 5¼" or 3½" floppy diskettes

Rush me DESQview 2.0! Today!

No. of Copies	Media 3½"/5¼"	Product	Retail Price ea.	Total
		DESQview 2.0	$129.95	$
		Shipping & Handling USA	$ 5.00	
		Outside USA	$ 10.00	$
		Sales Tax (CA residents)	6.5%	$
			Amount Enclosed	$

Payment: ☐ Visa ☐ MC ☐ AMEX ☐ Check

Credit Card: Valid Since ____/____ Expiration ____/____

Card Number: | | | | | | | | | | | | | | | | |

Credit Card Name ____

Shipping Address ____

City ____ State ____ Zip ____ Telephone ____

Mail to: Quarterdeck Office Systems, 150 Pico Boulevard, Santa Monica, CA 90405
NOTE: If you own DESQview call us for a special upgrade offer, or send in your DESQview registration card and we'll send you upgrade information.

DDJ 8

DESQview™
Quarterdeck

Quarterdeck Office Systems • 150 Pico Boulevard, Santa Monica, CA 90405 • (213) 392-9851

CIRCLE 284 ON READER SERVICE CARD

The most brilliant breakthrough in SQL technology since SQL.

XQL is a dramatic step forward in the history of SQL. It's the one unique SQL solution that helps programmers break through to even higher levels of productivity. Powerful yet easy to use, XQL minimizes your coding time and lets you focus on building better applications.

XQL extends the power of Btrieve, SoftCraft's high-performance file manager, by allowing access to multiple records at a time. It frees your application from physical file characteristics by providing true relational capabilities with data independence, data descriptions, data integrity and security.

XQL's three interface levels are a major advance in SQL technology. The first two levels, XQL primitives for maximum efficiency or full SQL statements for maximum convenience, are callable subroutines from BASIC, Pascal and C. The third level lets you enter SQL statements interactively without ever having to write a program.

XQL's extensive DBMS features let you access data by name. Field order is independent of physical location within the Btrieve record. Only records that pass your restrictions are returned—in the sort order you specify. Fields can be computed from other fields or constants. And you can manipulate composite records built from multiple, joined Btrieve files.

XQL offers all the performance and reliability you've come to expect from Btrieve, including LAN support, fault tolerance, comprehensive documentation and expert technical support for trouble-free software development. Plus, you never pay royalties on your XQL applications.

Put the latest innovation in SQL technology to work for you. Contact SoftCraft.

SoftCraft
A NOVELL COMPANY
P.O. Box 9802, #917
Austin, Texas 78766
(512) 346-8380 Telex 358 200

XQL, $595; Btrieve, $245; multiuser Btrieve, $595. XQL requires Btrieve and PC-DOS or MS-DOS 2.X or 3.X. XQL is a trademark and Btrieve is a registered trademark of SoftCraft, Inc.

CIRCLE 113 ON READER SERVICE CARD

LETTERS

Dvorak Responds

Dear *DDJ*,

I was greatly amused (Swaine's Flames, June 1987) by the rambling cockroach-like insect that supposedly jumped out of Swaine's computer and onto the keyboard late one night while Mike was puking his guts out after having been to one of those dubious and fashionable sushi bars found far too often south of Palo Alto where fresh fish is a rarity and where the wallpaper, when examined closely, turns out to be posted health department notices of violation.

No doubt was left in the reader's mind that the whole incident was a hallucination of grand proportion or (and we all know that this is the case) the column was a gimmicky idea possibly dreamed up while suffering the ill effects of poisoning and designed to solve one of two problems. They are: 1) how to make a point and blame someone (or something) else for making the point; 2) how to avoid the ravages of mean copy editors. Since the second point only applies to the profession of writing, one must assume that the first point applies.

Now obviously some assertion I made sometime when and who knows where struck a nerve inside the *Dr. Dobb's* editors cage. Thus the essay by the roach about me. Unfortunately, for Mike and the crew, my point concerning outspokenness was only reaffirmed by the technique of using an imagined insect to express an opinion. It surely wasn't done to avoid being nasty but only done to avoid confrontation. That's the problem (if there is a problem), in my opinion, with *Dr. Dobb's*—a certain inexplicable timidity mistakenly interpreted by some as politeness.

Now I advise you to look at my current benefactor and publisher, *PC Magazine*. The commentary in there is sometimes downright rude. Editors even have the gall to make Editors Choices, a concept that flies in the face of advertising theory. Yet the magazine is as fat as imaginable and growing in circulation like nothing I've ever seen. Why? Because today's reader is bored and prefers to read copy written with a sharp edge. There is too much dreary stuff out there taking up too much space in our lives.

The editors of *Dobb's*, I know for a fact, have this same ability to produce enjoyable vitriol but prefer to be well liked instead. The out-of-place and gratuitous swipe at me by the verbose roach claiming I was an incarnation of Max Headroom is a good example of the kind of nastiness that lurks within the *Dobb's* staff.

While the entire piece was a gem of creativity insofar as finding some unusual way to couch an essay, I still maintain that using a bug to make a complaint is symbolic not of dignity but of a certain kind of shyness that weakens the impact of the printed word. Then again, the whole story may be true. If so, I advise Swaine to vacuum more often.

John C. Dvorak
PC Magazine
Ziff-Davis Publishing Co.
One Park Ave.
New York, NY 10016

Michael Swaine replies:

Thanks, John. It's an honor to get a lesson from the leading practitioner of no-nonsense, outspoken, sharp-edged computer journalism. By the way, I enjoyed your recent *PC Magazine* column tracing the origin of the word *nerd* back to Dr. Seuss.

XOR Chains

Dear *DDJ*,

David Cortesi's article on XOR chains (June 1987) is an excellent example of isolating information in modules.

I disagree with one of his conclusions, though: the *Insert* procedure can do the same damage to a list as the *Delete* procedure can. When two different *Scans* each insert an item into the same position in the list, the list structure is destroyed.

For example: let *Scans P* and *Q* each have item *A* as *next* and item *B* as *prev*. Then *Insert (C,P)* will insert an item *C* between *A* and *B* in the list. *Scan P* will correctly show item *A* as *next* and item *C* as *prev*, but *Scan Q* will still treat items *A* and *B* as adjacent items. Calling *Insert (D,Q)* now will cause *Insert* to try, incorrectly, to XOR *B* to the link in *A* as if they were still adjacent items. This XOR operation and the XOR of *A* to the link in *B* will leave *A* and *B* with invalid links.

This flaw in *Insert* can be corrected by having *Insert*

Harold applies algorithmic logic to everyday situations

To Develop Tomorrow's Applications, You Need The DOS Of The Future.

WENDIN-DOS™

THE MULTITASKING, MULTIUSER MS-DOS REPLACEMENT

- RUNS ON ANY IBM-PC, XT, AT, 80386, OR TRUE COMPATIBLE
- RUNS MS-DOS PROGRAMS AND USES THE MS-DOS FILE SYSTEM
- FEATURES FILE SHARING AND FILE LOCKING
- PROVIDES TRUE CONCURRENT MULTITASKING AND TASK SWITCHING
- SUPPORTS MULTIPLE TERMINALS WITH NO EXTRA SOFTWARE REQUIRED
- FEATURES A USER-CONFIGURABLE WINDOWING INTERFACE
- ALLOWS ADDRESSING OF EXTENDED MEMORY
- FEATURES A FILE PERMISSION SYSTEM
- SUPPORTS THE MS-DOS COMMAND LANGUAGE AND EXTENDS IT WITH COMMANDS LIKE PROTECT, PRIV, SPAWN, AND KILL

ONLY $99 — DISKETTES AND MANUAL INCLUDED

And You Need The Tools To Get The Job Done Right . . .

THE WENDIN-DOS™ APPLICATION DEVELOPER'S KIT

- GIVES ACCESS TO OVER 80 NEW SYSTEM SERVICES SUPPORTED BY WENDIN-DOS
- INCLUDES LIBRARIES WRITTEN IN ASSEMBLY FOR ACCESS FROM HIGH-LEVEL LANGUAGES LIKE C
- ALLOWS USER PROGRAMS TO ACCESS SYSTEM SERVICES FOR PROCESS CONTROL, SWAPPING, MEMORY MANAGEMENT, RECORD AND FILE MANAGEMENT, AND CONCURRENT I/O.
- PROVIDES AN I/O SUBSYSTEM THAT SUPPORTS CONCURRENT I/O OPERATIONS ON ANY OF THREE DIFFERENT LEVELS OF ACCESS: PHYSICAL, LOGICAL, AND VIRTUAL.

ONLY $99 — DISKETTE, SOURCE CODE AND MANUAL INCLUDED

TO ORDER WENDIN-DOS or THE WENDIN-DOS APPLICATION DEVELOPER'S KIT — CALL (509) 624-8088 or

SIMPLY SEND A BRIEF, WRITTEN REQUEST FOR INFORMATION ABOUT ANY WENDIN PRODUCT TO:

PLATINUM SERIES
FLEXIBLE DISKETTES

Wendin, Inc.
P.O. Box 3888
Spokane, WA 99220-3888

WENDIN®

We will send you an exciting full color catalog and a FREE product line demo diskette, while supplies last, compliments of Syncom Technologies, Inc., and Wendin, Inc. (509) 624-8088.
(System hardware recommendation — minimum 512K and for multitasking a machine with at least the computing power of an IBM-AT.)

WENDIN®
BOX 3888
SPOKANE, WA 99220-3888
Working beyond the horizon to develop the operating systems of tomorrow

 (509) 624-8088

DEALER INQUIRIES WELCOME
Foreign orders inquire about shipping.
Domestic orders add $6.00/ 1st item, $1.00 each additional item for shipping, handling, and insurance. We accept Visa/MC, American Express, C.O.D., and Bank Drafts drawn on U.S. Banks.
Washington residents add 7.8% sales tax.

MS is a trademark of Microsoft. PC-DOS is a trademark of IBM.

CIRCLE 112 ON READER SERVICE CARD

check if the elements in the *Scan* are still adjacent before inserting the new item.

Andrew Ginter
208-21 Ave. NW
Calgary, AB
Canada T2M 1J3

Dear *DDJ*,
In his article Mr. Cortesi mentions clearing a register in assembly language by XORing it with itself. It also might be worth mentioning that you can swap two fields quickly without using a work area via use of the XOR command thrice:

```
XC   FIELD1,FIELD2
XC   FIELD2,FIELD1
XC   FIELD1,FIELD2
```

This will put the contents of *FIELD1* into *FIELD2* and vice versa. Note that the fields must be of equal length. I'm sure that this also works just as quickly in other languages because typically Boolean algebra commands are built-in commands in almost any processor.

Jack Gillette
30 Harvest Rust Ct.
St. Charles, MO 63303

Dear *DDJ*,
David E. Cortesi's fine article presented your readers with some valuable information. However, the algorithms he describes are not the simplest possible because he uses an unnecessarily complicated logical tautology. The relation he used in his article was:

(A XOR B XOR C) XOR B XOR C = A

This relation is actually one case of a more general relationship, namely:

(X_1 XOR X_2 XOR . . . XOR X_N) XOR
X_2 XOR . . . XOR X_N = X_1

I'm not sure if this general relation is of much value, but its simplest form is more suitable for the task of storing both front and back pointers in a single field. The simplest form merely says:

(A XOR B) XOR B = A

Nowhere in the algorithms that Mr. Cortesi describes does he require the fact that his link field stores the address of the current item in the list. With this in mind, his implementation can be improved with the following changes:

1. For an item in the list, if A is the address of its predecessor and B is the address of its successor, then set the single link word W of the item to:

W = A XOR B

You can now recover B as (W XOR A) or A as (W XOR B).
2. The stepping routines *GoFwd* and *GoBak* should be modified to remove the use of the current item address. For example:

```
Void GoFwd(s) struct Scan *s;
struct Item *i;
i =    s->prior Xor
s->next->link;
s-     >prior = s->next;
s-     >next = i;
       }
```

3. The insertion and deletion routines *Insert* and *Delete* should be modified in a similar fashion. For the sake of accuracy (and to correct some minor errors in the original article), I have given each of these in full in Example 1, right.

The changes described in Example 1 amount to a savings of one XOR operation per call to any of these four routines. Although this may not seem like much, it can make a significant difference in the time required to process a large list. Of course, the effect on code size is minimal.

Mr. Cortesi correctly asserts that traversing a data structure linked in this fashion requires the addresses of two logically adjacent items. However, this does not lock you into a list-type structure. I am submitting a follow-up article entitled "The XOR Chain Revisited" that describes how such links can be used effectively in manipulating tree-type data structures.

Bennette R. Harris
231 S. Janesville St.
Whitewater, WI 53190

Mr. Harris' article begins on page 36 in this issue. —eds.

68000 Dynamic Halt

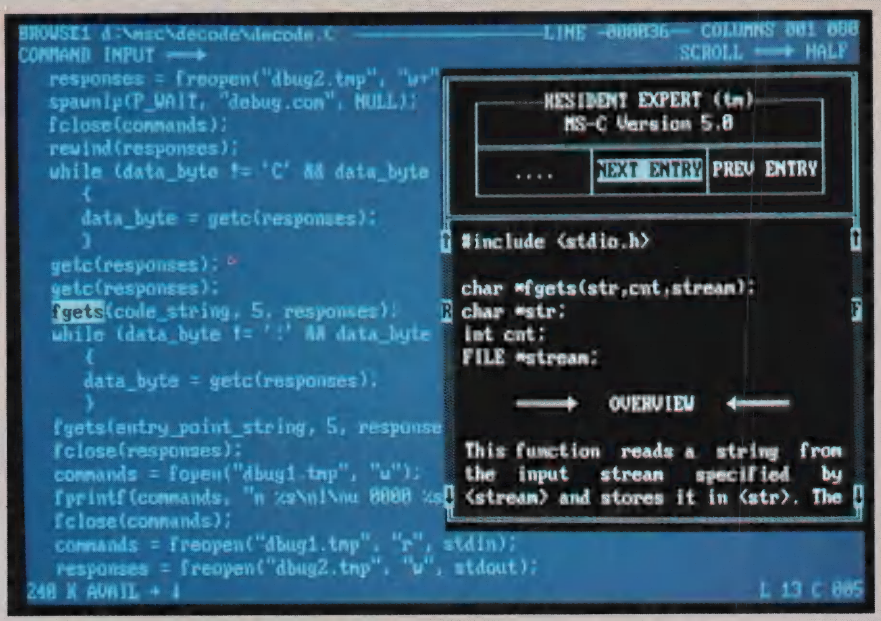
Dear *DDJ*,
The *FOR TOP* to *BOT* loop in Brian Anderson's Viewpoint (June 1987) will only reach *BOT* in the unlikely event that *TOP* = *BOT*! Else we have yet another fine Dynamic Halt (aka endless loop). *CLR.B 0(A0,D0)* does not alter value *y D0*. So did your typesetter omit *ADDQ.W #1, D0* before the loop branch? Brian would never do such a thing (and yet his line numbers seem

(continued on page 138)

```
void Insert(i,s)
struct Item *i;
struct Scan *s;
{
if (AtHead(s))
  s->base->head = i;
else
  s->prior->link Xor= (i Xor s->next);
if (AtTail(s))
  s->base->tail = i;
else
  s->next->link Xor= (i Xor s->prior);
i->link = s->prior Xor s->next;
s->prior = i;
}
void Delete(s) struct Scan *s;
struct Item *i;
if (AtTail(s)) return;
i = s->next->link Xor s->prior;
if (AtHead(s))
  s->base->head = i;
else
  s->prior->link Xor= (s->next Xor i);
if (i == Nil)
  s->base->tail = s->prior;
else
  i->link Xor= (s->next Xor s->prior);
DropItem(s->next);
s->next = i;
}
```

Example 1: *Modified insertion and deletion routines for "The XOR Chain"*

RESIDENT EXPERT Pop-up Reference Guides...



Try One And Get Our MS-DOS/PC-DOS Guide Absolutely FREE!

THE POP-UP REFERENCE REVOLUTION BEGINS

How much development time could you save if you never had to open another PC language or technical reference manual again? What if you could just point at a compiler keyword, assembly instruction, or function name *on your screen* and with a keystroke have complete, authoritative information about language syntax, operands, parameters, examples, and much more?

INTRODUCING THE RESIDENT EXPERT SYSTEM

A growing library of comprehensive, disk resident reference guides about the PC *and* your favorite PC languages. All available instantly through our unique memory resident pop-up access system.

VIRTUALLY EVERYTHING YOU NEED TO KNOW

Each of our *Compiler Reference Guides* contains virtually everything you need to know to program with your *preferred implementation* of your favorite language. Language syntax, all library functions, complier directives, and error codes are thoroughly documented.

Our *PC Programmer's Reference Guide* documents every PC (and AT) processor instruction and every BIOS and DOS service interrupt. You'll also find tables of keyboard codes, line drawing, ASCII, and IBM character sets, and much more.

THE SPECIALIST'S LIBRARY

Your compiler is unique. That's why our reference guides are *specialized*...each one designed for a particular vendor's language implementation.

Free With Any Purchase!

Our Companion Pop-up Guide To MS-DOS 3.2/PC-DOS 3.3

Limited Time Offer

QUICK DRAW ACCESS SYSTEM

Point-and-shoot...just place the cursor over any term on your screen. Chances are we've got it fully detailed in one of our data bases.

Fully cross indexed...if the instruction or library function you're using isn't quite right, our related topics cross index can help you find a better one.

Multiple volumes on line...you can have one or a dozen of our pop-up reference guides on line...a complete library available instantly.

THE INFORMATION YOU NEED...WHERE YOU NEED IT

Our pop-up shell varies its size and shape dynamically, only taking as much space on your screen as it needs and it *never* covers your working area. You can see your work and our reference data at the *same* time.

A COMPLETE LIBRARY...STILL ONLY A BEGINNING

At Santa Rita, our commitment is to provide the most accurate, extensive selection of PC language reference materials available. If you don't see one of our guides for your favorite language or compiler listed below don't worry, we're probably working on it!

PC Programmer's Reference Guide . . . $59.00
(with Assembly Language Guide)

Borland Turbo C *(1.0)* 59.00
Borland Turbo Pascal *(3.0 and below)* 59.00
(with Graphics & Numerical Methods Toolbox)
Borland Turbo Prolog *(1.1 and below)* 59.00
(with Prolog Toolbox)
Lattice C Compiler *(3.2 and below)* 59.00
Mark Williams LetsC *(4.0 and below)* 59.00
Microsoft C Compiler *(5.0 and below)* 59.00

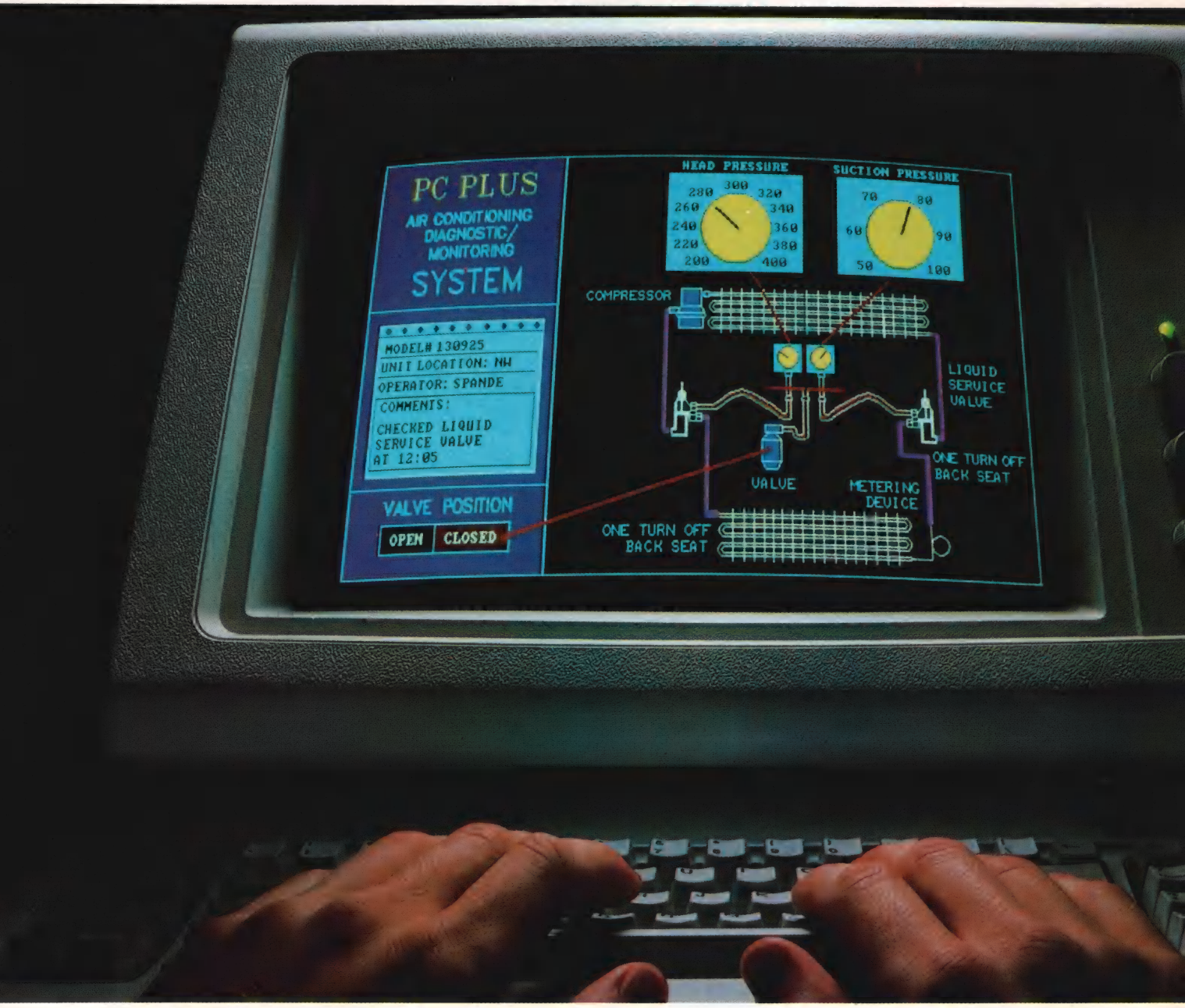
SantaRita

For the location of your nearest Santa Rita Software dealer, or to order direct, call us at 1-214-727-9217. We'd like to hear from you.

Santa Rita Software
1000 E. 14th Street, Suite 365
Plano, Texas 75074

The RESIDENT EXPERT System

Texas Instruments has system developers need.



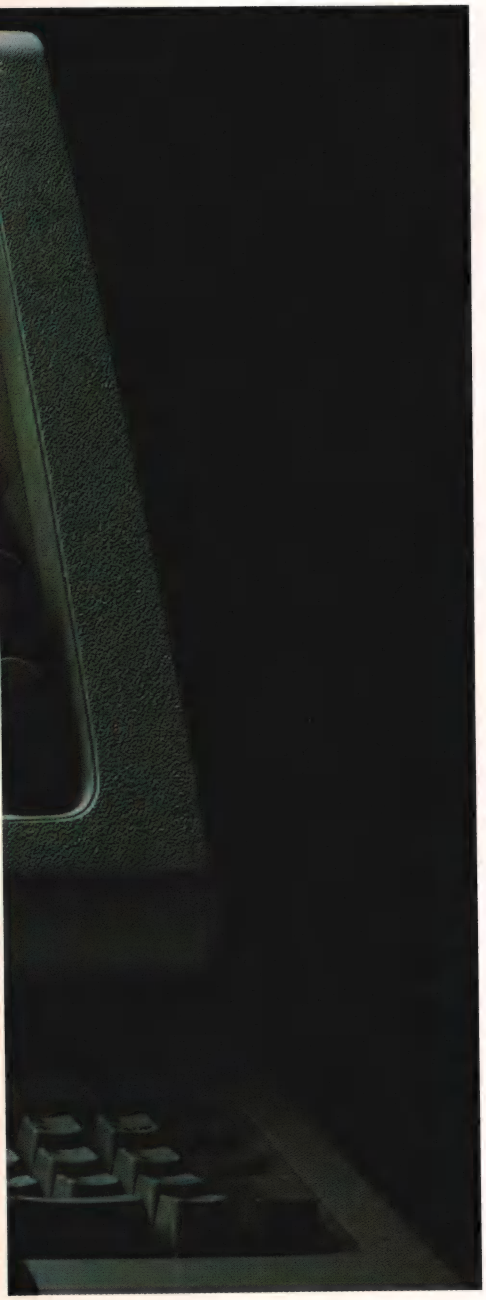
"Personal Consultant™ Plus... offers a very fine expert system development and delivery tool that already has a proven record with end-users."

— *Susan Shepard, AI Expert*

Personal Consultant Plus 3.0 Standard Features
- Frames, rules, meta rules and procedures
- Forward/backward chaining
- Confidence factors
- Regression testing and rule tracing
- End-user explanation facilities
- Graphics image capture and display
- Interfaces to dBase™, Lotus 1-2-3™, DOS files, .EXE or .COM programs, "C"
- Complete LISP development environment
- 2-megabyte expanded/extended memory support
- Mouse support
- Context sensitive help
- "Getting Started" tutorial-style manual

Personal Consultant Images
- Optional add-on package to PC Plus (3.0)
- Allows integration of "active images" into

what serious expert Power tools.



knowledge bases
- Interactive dials, gauges, forms and selection images
- Multiple images can be combined on same screen
- "Getting Started" tutorial-style manual

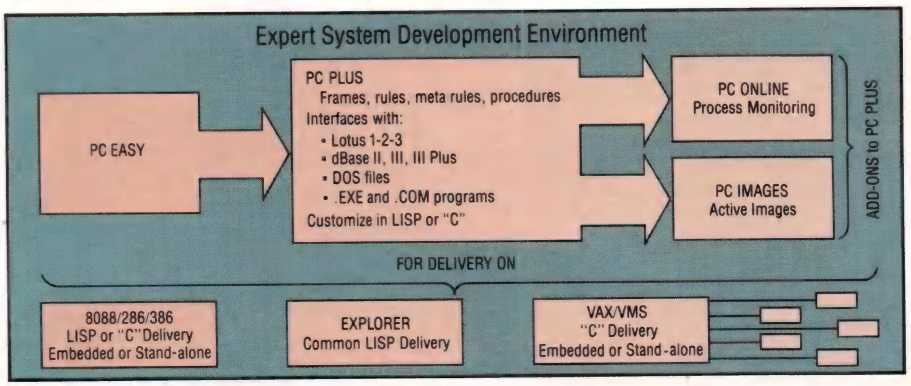
Personal Consultant Online
- Optional add-on package for PC Plus (3.0)
- ONLINE expert systems that interact directly with process data
- Multiple interfaces to data acquisition and analysis programs
- Knowledge base synchronization with process data
- Functions for historical and predicted trends
- Special user interface/reporting capabilities
- "Getting Started" tutorial-style manual

Among all the expert system development tools available for personal computers today, none deliver the power and flexibility of TI's Personal Consultant series.

Personal Consultant Easy is ideal for getting started, and is upwardly compatible with the higher functionality of PC Plus. For experienced developers, Personal Consultant Plus and its optional add-on enhancements, Online and Images, were designed to help solve a broader range of complex problems.

package helps deliver expertise that is "online all the time."

Application delivery as flexible as the tools themselves.
Delivery can be in LISP for flexibility, or "C"* for maximum speed and portability. Our "C" options support either stand-alone or "embedded" knowledge bases. Options are available for DOS-based PCs, TI's Explorer, and DEC's VAX™ line of multi-user minis running under VMS™.



Personal Consultant Plus. Full power for an affordable price.
At $2,950, PC Plus has proven to be one of the richest and most flexible problem-solving tools available for the development of complex knowledge-based systems. Designed to take advantage of today's more powerful 286/386 DOS-based computers, or TI's Explorer™ Symbolic Processing System, the new 3.0 version of PC Plus provides powerful standard features and a continuing growth path with the addition of either PC Images or PC Online, or both.

Personal Consultant Images. Picture an expert system with interactive graphics.
At $495, PC Images enables developers to create knowledge-based applications that incorporate complex graphical "active images." User-interactive dials, gauges, forms and selection images provide a more exciting visual data input and output style.

Personal Consultant Online. The expert system as part of the process.
At $995, PC Online allows the developer to design expert systems which interact directly with process data, as opposed to input from a human operator. Designed for intelligent process monitoring applications, this optional

"Texas Instruments has done more than any other company to educate people about AI, to popularize it, and to make useful AI tools available at reasonable prices."
— Jim Seymour, PC Magazine.

Technical support, training courses and Knowledge Engineering Services are available for the Personal Consultant products. If you have a question about any of our expert system power tools, we have the answer.

Pick up the phone and gain a powerful advantage.
Call **1-800-527-3500** for technical overviews of our products and a PC Plus case histories brochure which details how our power tools are being put to work today.

36106

*Available 4Q 1987.

TEXAS INSTRUMENTS

How Many Ways Can You Draw a Circle?

by James F. Blinn

A comprehensive roundup of circle-drawing algorithms

I like to collect things. When I was young I collected stamps; now I collect empty margarine tubs—and algorithms for drawing circles. Because this is an algorithms issue, I will restrain myself from talking about stamps or margarine tubs; instead, I'll show you my album of circle-drawing algorithms.

It's traditional at this point in any discussion of geometry to drag in the ancient Greeks and mention how they considered the circle to be the most perfect shape. Even though a circle is such an apparently simple shape, it is interesting to see how many essentially different algorithms you can find for drawing the Greek's favorite curve.

I will be brief about some pretty obvious techniques to leave space to play with the more interesting and subtle techniques. Note that many of these algorithms might be ridiculously inefficient but are included to pad out the article. (OK, OK—they're actually included for completeness.)

I'm not sure where I first heard of some of these. I will cite inventors where known, but let me just thank the world at large in case I've missed anybody.

A word about the programming language I use—I am not using any formal algorithm display language here. These algorithms are meant to be read by human beings, not computers, so the language I present is a mishmash of several programming constructs that I'm sure will be perfectly clear to you.

The collection can be categorized by the two types of output—line endpoints and pixel coordinates. This comes from the general dichotomy of curve representation—parametric vs. algebraic.

Line Drawings

First, I'll look at line output. All the algorithms in this section operate in floating point and generate a series of x,y points on a unit-radius circle centered at the origin. You then play connect the dots.

1. Trigonometry

Evaluate sin and cos at equally spaced angles:

```
MOVE(1,0)
FOR DEGREES=1 to 360
    RADIANS=DEGREES*2*3.14159/360.
    DRAW(COS(RADIANS),SIN(RADIANS))
```

This has to evaluate the two trig functions at each loop, ick.

James F. Blinn, 195 South Wilson, Apt. 8, Pasadena, CA 91106. Dr. Blinn has been actively involved in computer graphics for the past 19 years. Since 1978 he has been at the Jet Propulsion Laboratory of the California Institute of Technology, producing animations depicting various space missions. He also produced computer graphics effects for the PBS series "COSMOS." He currently teaches courses in computer graphics at Cal Tech and at the Art Center College of Design in Pasadena.

This is a revised version of an article published in the August 1987 issue of IEEE Computer Graphics and Applications.

2. Polynomial Approximation

You can get a fair approximation of a circle by evaluating simple polynomial approximations to sin and cos. The first ones that come to mind are the Taylor's series:

$$\cos a \approx 1-(\frac{1}{2})a^2+(\frac{1}{24})a^4$$
$$\sin a \approx a-(\frac{1}{6})a^3+(\frac{1}{120})a^5$$

These require fairly high-order terms to get very close.

A better approach is to fit lower-order polynomials to the desired endpoints and end slopes. This is effectively what happens with various commonly used Bezier curves—for example, the four control points (1,0), (1,0.552), (0.552,1), (0,1) describe a good approximation to the upper-right quarter of a circle. You can get the other three quadrants by rotating the control points.

When transformed to polynomial form, the first quadrant is:

$$x(t) = 1-1.344t^2 + .344t^3$$
$$y(t) = 1.656t-.312t^2 -.344t^3$$

with the parameter t going from 0 to 1.

```
MOVE(1,0)
FOR  T = 0 TO 1 BY .01
     X = 1 + T*T*(-1.344+T*.344)
     Y = T*(1.656-T*(.312+T*.344)
     DRAW(X,Y)
```

This makes a pretty good circle—the maximum radius error is about 0.0004 at $t=0.2$ and $t=0.8$.

3. Forward Differences

Polynomials can be evaluated quickly by the technique known as forward differences. Briefly, for the polynomial:

$$f(t) = f_0 + f_1t + f_2t^2 + f_3t^3$$

if you start at $t=0$ and increment by equal steps of size δ, the forward differences are:

$$\Delta f = f_1\delta + f_2\delta^2 + f_3\delta^3$$
$$\Delta\Delta f = 2f_2\delta^2 + 6f_3\delta^3$$
$$\Delta\Delta\Delta f = 6f_3\delta^3$$

Then, for polynomials stepping in units of 0.01:

```
X=1; DX=-.000134056; DDX=-.000266736; DDDX= .000002064
Y=0; DY= .016528456; DDY=-.000064464; DDDY=-.000002064
MOVE(X,Y)
FOR  I=1 TO 100
     X=X+DX;  DX=DX+DDX;  DDX=DDX+DDDX
     Y=Y+DY;  DY=DY+DDY;  DDY=DDY+DDDY
     DRAW(X,Y)
```

Trust me, I'm a doctor. If you don't believe it, look up forward differences on page 328 of Newman and Sproull—I'm not going to do all the work here.

Notice the number of significant digits in the constants. It might seem as though that many digits would require double precision, but in practice the accumulated round-off error using single precision is less than the error due to the polynomial approximation.

4. Incremental Rotation

Let's back off from the approximation route and try another approach. Start with the vector (1,0) and multiply it by a one-degree rotation matrix each time through the loop:

DRAWING CIRCLES
(continued from page 19)

```
DELTA = 2*3.14159/360.
SINA=SIN(DELTA)
COSA=COS(DELTA)
X=1; Y=0
MOVE(X,Y)
FOR I=1 TO 360
      XNEW = X*COSA - Y*SINA
      Y = X*SINA + Y*COSA
      X = XNEW
```

5. Extreme Approximation

If the incremental angle is small enough, you can make the approximations $\cos a = 1$ and $\sin a = a$. The number of times through the loop is $n = 2\pi/a$, or conversely, the angle is $a = 2\pi/n$, depending on which you want to use as input:

```
A=.015; N=2*3.14159/A
X=1; Y=0
MOVE(X,Y)
FOR I=1 TO N
      XNEW = X - Y*A
      Y = X*A + Y
      X = XNEW
      DRAW(X,Y)
```

But there's a problem. Each time through the loop, you are forming the product:

$$(x_{new}, y_{new}) = (x_{old}, y_{old}) \begin{pmatrix} 1 & a \\ -a & 1 \end{pmatrix}$$

The matrix is almost a rotation matrix, but its determinant equals $1+a^2$. This is bad. It means that the running (x,y) is magnified by this amount on each iteration, so what you get is a spiral that gets bigger and bigger. How to fix this?—introduce a bug into the algorithm.

6. Unskewing the Approximation

Because vector multiplication and assignment don't occur in one statement, you had to calculate y carefully, using the old value for x. Suppose you were dumb and did it the naive way:

```
A=.015; N=2*3.14159/A
X=1; Y=0
MOVE(X,Y)
FOR I=1 TO N
      X = X - Y*A
      Y = X*A + Y
      DRAW(X,Y)
```

Now, what is the effect of this? Really what you get is:

$$x_{new} = x_{old} - y_{old} a$$

$$y_{new} = x_{new}\, a + y_{old} = x_{old}\, a + y_{old}\,(1-a^2)$$

In other words:

$$(x_{new}, y_{new}) = (x_{old}, y_{old}) \begin{pmatrix} 1 & a \\ -a & 1-a^2 \end{pmatrix}$$

This matrix has a determinant of 1, and there is no net spiraling effect. What you get is actually an ellipse that is stretched slightly in the northeast-southwest direction and squeezed slightly in the northwest-southeast direction. The maximum radius error in these directions is approximately $a/4$.

Now comes the really interesting part. Because you can start out with any vector, let's try (1000,0). Now, cleverly select a to be an inverse power of 2 and the multiplication becomes just a shift—for example, a value of $a = 1/64$ is just (shift right 6) and generates the circle in about 402 steps. So, you can do all this just with integer arithmetic and no multiplication. This is how you used to draw circles quickly—and in fact do rotation incrementally—before the age of hardware floating point and even hardware multiplication. (This was probably invented by Ivan Sutherland.)

7. Rational Polynomials

Another polynomial tack can be taken by looking in your hat and pulling out the following rabbit:

if:

$$x = (1-t^2)/(1+t^2)$$
$$y = 2t/(1+t^2)$$

then:

$$x^2 + y^2 = 1$$

no matter what t is (or identically, as mathematicians would say). Running t from 0 to 1 gives the upper-right quadrant of the circle. You can again evaluate these poly-

EVEN MORE POWER AND FLEXIBILITY

BRIEF 2.0

Users and industry press alike have unanimously proclaimed BRIEF as the best program editor available today. Now, the best gets better, with the release of BRIEF 2.0.

Straight from the box, BRIEF offers an exceptional range of features. Many users find that BRIEF is the only editor they'll ever need, with features like real, multi-level Undo, flexible windowing and unlimited file size. But BRIEF has tremendous hidden power in its exclusive macro language. With it, you can turn BRIEF into your own custom editor containing the commands and features you desire. It's fast and easy.

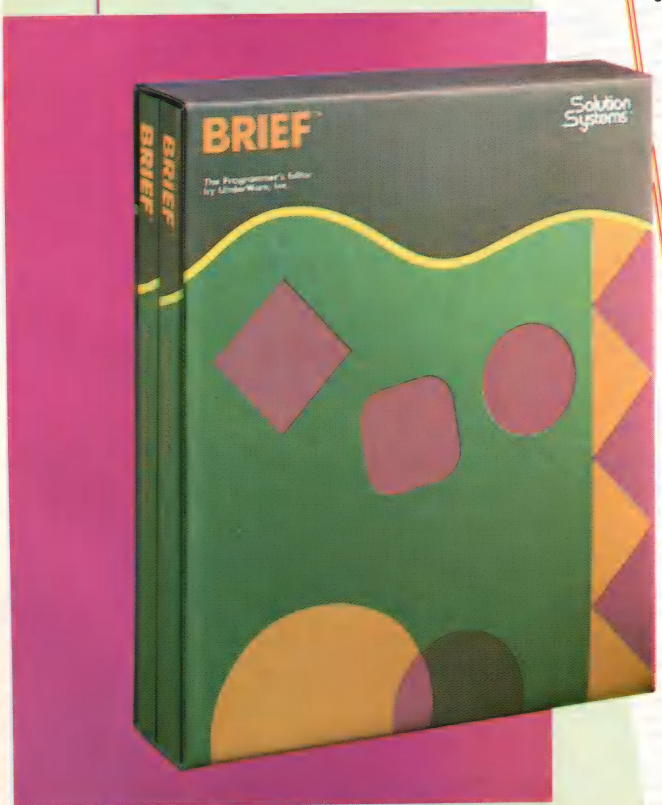
Jerry Pournelle, columnist for BYTE magazine summed it all up by saying BRIEF is, "**Recommended. If you need a general purpose PC programming editor, look no further.**" His point of view has been affirmed by rave reviews in C JOURNAL, COMPUTER LANGUAGE, DR. DOBB'S JOURNAL, DATA BASED ADVISOR, INFOWORLD AND PC MAGAZINE.

One user stated "BRIEF is one of the few pieces of software that I would dare call a masterpiece." Order BRIEF now and find out why. **BRIEF 2.0 is just $195. If you already own BRIEF, call for upgrade information.**

TO ORDER CALL: 1-800-821-2492 (in MA call 617-337-6963)

As always, BRIEF comes with a 30 day money-back satisfaction guarantee.

Solution Systems™
541 Main Street
Suite 410D
So. Weymouth, MA 02190
(617) 337-6963



Look at these BRIEF 2.0 enhancements!

Main Features:

- All new documentation with tutorials on basic editing, regular expressions **and** the BRIEF Macro Language.
- Setup program for easy installation and configuration. (Requires no knowledge of the macro language)
- Increased speed for sophisticated operations like Undo and Regular Expression Search.
- Expanded regular expressions, with matching over line boundaries.
- More block types, with marking by character, line or column.
- Command line editing (move cursor, add and delete characters, specify command parameters).
- Support for more programming languages.
- Optional borderless windows.
- Enhanced large display support, including wider displays.
- Reconfigurable indenting for C files (supports most indenting styles).

Plus the basic features that made BRIEF **SO** popular!

Basic Features:

- Full multi-level Undo
- Windows
- Edit many files at once
- File size limited only by disk space
- Automatic language sensitive indentation

Requires an IBM PC or compatible with at least 192K RAM.

BRIEF is a trademark of UnderWare, Inc.

Solution Systems is a trademark of Solution Systems.

CIRCLE 142 ON READER SERVICE CARD

nomials by forward differences, stepping t in increments of 0.01, and get:

```
X=1; DX=-.0001; DDX=-.0002
Y=0; DY= .02
W=1; DW= .0001; DDW= .0002
MOVE(X,Y)
FOR I=1 TO 100
      X=X+DX; DX=DX+DDX
      Y=Y+DY
      W=W+DW; DW=DW+DDW
      DRAW(X/W,Y/W)
```

Note that this is not an approximation like the last few tries were. It is exact—except for round-off error. Even round-off error can be removed, either by calculating the polynomials directly or by scaling all numbers by 10,000 and doing it with integers. (The division x/w must still be done in floating point.)

This one has always amazed me—effectively, you get to evaluate two transcendental functions exactly with only a few additions. What's the catch? It's an application of the No Free Lunch theorem—you don't get to pick the angles. If you watch the points, you see that they are not equally spaced around the circle. In fact, as t goes to infinity, the point keeps going counterclockwise but slows down, finally running out of juice at (–1,0). If you go backward to minus infinity, the point goes clockwise, finally stopping again at (–1,0). (Yet more evidence that $-\infty = +\infty$.) To draw a complete circle, you are best advised to run t from –1 to +1, which draws the whole right half, and then mirror it to get the left half.

8. Differential Equations

An entirely different technique is to describe the motion of (x,y) dynamically. Imagine the point rotating about the center as a function of time t. The position, velocity, and acceleration of the point will be:

$$(x,y) = (\cos t, \sin t)$$
$$(x', y') = (-\sin t, \cos t) = (-y, x)$$
$$(x'', y'') = (-\cos t, -\sin t) = (-x, -y)$$

You can cast these into differential equations and use several numerical integration techniques to solve them.

The dumbest one, Euler integration, is just:

$$x_{new} = x_{old} + x'_{old}\,\Delta t = x_{old} - y_{old}\,\Delta t$$
$$y_{new} = y_{old} + y'_{old}\,\Delta t = y_{old} + x_{old}\,\Delta t$$

This looks a lot like algorithm 5 and has the same spiraling-out problem. You can generate better circles by using better integration techniques. My two favorites are the "leapfrog" technique and the Runge-Kutta technique.

Leapfrog calculates the position and acceleration at times:

$$t,\ t+\Delta t,\ t+2\Delta t, \ldots$$

but calculates the velocity at times halfway between them:

$$t+\tfrac{1}{2}\Delta t,\ t+\tfrac{3}{2}\Delta t, \ldots$$

Advancing time one step then looks similar to Euler with just the evaluation times offset:

$$x_{t+\Delta t} = x_t + x'_{t+\frac{1}{2}\Delta t}\,\Delta t$$

$$x'_{t+\frac{3}{2}\Delta t} = x'_{t+\frac{1}{2}\Delta t} + x''_{t+\Delta t}\,\Delta t$$

(with similar equations for y). The position and velocity leapfrog over each other on even/odd half-time steps, so you have to keep separate variables for the velocities x' and y' The code has a lot in common with algorithm 6 and probably for good reason:

```
X=1 ; Y=0
VX=-SIN(DT/2); VY=COS(DT/2)
MOVE(X,Y)
FOR I = 1 TO N
      X = X + VX*DT      "update posn"
      Y = Y + VY*DT
      VX = VX - X*DT     "update veloc, AX=-X"
      VY = VY - Y*DT     "              AY=-Y"
      DRAW(X,Y)
```

Runge-Kutta is a slightly involved process that takes a fractional Euler step, reevaluates the derivatives there, applies the derivative at the original point, steps off in this new direction, generally screws around, and finally takes some average of all these to get the new time step. Plugging the differential equation into the formulas and simplifying requires about a page of algebra. You can look up the actual equations; they're not incredibly complicated, but their derivation is "beyond the scope" of almost all numerical analysis textbooks I have seen.

One advantage of Runge-Kutta is that it finds the position and velocity at the same time step, so for circles you can generate x and y with the same computation. Another advantage is that it comes in second-order, third-order, fourth-order, and so on versions for higher orders of precision than does leapfrog.

Plugging in for second-order RK, the ultimate result is:

$$x_{new} = x_{old}(1 - \tfrac{1}{2}\Delta t^2) + y_{old}(-\Delta t)$$

$$y_{new} = x_{old}(\Delta t) + y_{old}(1 - \tfrac{1}{2}\Delta t^2)$$

Does this look familiar? The third-order RK and another page of algebra leads to:

$$x_{new} = x_{old}(1 - \tfrac{1}{2}\Delta t^2) + y_{old}(-\Delta t + \tfrac{1}{6}\Delta t^3)$$

$$y_{new} = x_{old}(\Delta t - \tfrac{1}{6}\Delta t^3) + y_{old}(1 - \tfrac{1}{2}\Delta t^2)$$

Guess what fourth-order RK gives . . . you're right. I won't even bore you with the code.

"...a real UNIX System V for your 386 or 286-based PC!"

Microport System V/AT and Microport System V/386 are fully-licensed implementations of the certified AT&T UNIX System V Release 3 for personal computers.

Microport System V provides true multi-user and multi-tasking capabilities. It runs in protected mode and supports the entire address space available; up to sixteen megabytes on an AT and up to four gigabytes on an 80386. It supports all standard hard disks, multiple printers, multiple modems, 2 to 17 users on a single AT, and an unlimited number of users on an 80386.

Additional features:

- *Demand-paged, virtual memory management.*
- *Virtual consoles allow fifteen virtual windows of operation for UNIX applications and DOS windows under DOS Merge.*
- *Electronic mail, communications with uucp and CU.*
- *Dynamic buffer allocation.*
- *PC-DOS partitioning allows DOS and System V files to reside on the same hard disk (not required with DOS Merge).*
- *Shared libraries.*
- *80286 compatibility.*
- *Link kit to allow user installable device drivers.*
- *Menu-driven system administration.*
- *On-line help facility.*

Microport Sys Vision™ is a part of both System V/AT and System V/386, so you won't have to worry about system administration. Both are menu-driven and have context-sensitive help facilities. Sys Vision menus lets you perform all the system housekeeping and basic tasks without knowing the specific UNIX commands. It is menu-driven and has context-sensitive help facilities.

RUNTIME PACKAGE. The System V Runtime Package contains over 180 utility programs. It is one of the most comprehensive UNIX runtime systems currently available and includes the screen editor "vi" with user tutorials.

System V is sold with a two-user UNIX binary license. For an additional fee, users can obtain a kit to upgrade this license to an unlimited number of users.

80286:	List $ 199	**Ours $ 169**
80386:	List $ 199	**Ours $ 169**

UNLIMITED USER LICENSE KIT. Upgrades your System V Runtime for an unlimited number of users.

80286:	List $ 249	**Ours $ 209**
80386:	List $ 249	**Ours $ 209**

SOFTWARE DEVELOPMENT SYSTEM. This programming package includes an AT&T C Compiler that makes full and efficient use of your processor's extended instruction set. The code produced is among the most compact and fastest available.

Sdb, the source level debugger, allows display of the actual lines of C code being executed. Many useful tools such as SCCS (source code control system), make and ctrace are also provided.

80286:	List $ 249	**Ours $ 209**
80386:	List $ 499	**Ours $ 429**

TEXT PROCESSING SYSTEM. This package consists of the complete System V Release 2 Documentors Workbench (DWB). It includes both the new troff (device-independent troff) and the old troff (otroff). Drivers for the HP LaserJet Printer and APPLE's LaserWriter are optionally available.

80286:	List $ 199	**Ours $ 169**
80386:	List $ 199	**Ours $ 169**

COMPLETE SYSTEM V. Includes Runtime Package, Software Development Package and Text Processing Package.

80286:	List $ 549	**Ours $ 465**
80386:	List $ 799	**Ours $ 699**

DOS Merge. DOS Merge is a true multi-user and multi-tasking environment that supports concurrent use of both UNIX and DOS. With DOS Merge, you can develop software by combining DOS and UNIX programs and commands. You can even run both UNIX and DOS programs that use the same files. And DOS Merge 386 uses the full capabilities of the 80386's memory management so you can run multiple DOS programs and UNIX programs at the same time on dumb terminals. DOS Merge 386 comes in two versions: a 2-user version and an unlimited-user version.

80286:	List $ 149	**Ours $ 125**
80386 Unltd:	List $ 495	**Ours $ 429**
80386 2 User:	List $ 395	**Ours $ 345**

Now Available From Programmer's Connection

Microport Products

	List	Ours
System V/386 *Complete System*	799	699
Runtime System *(2 Users)*	199	169
Software Development System	499	429
Text Preparation System	199	169
System V/386 *Unlimited User License*	249	209
DOSMerge 386 *2-user System*	395	345
DOSMerge 386 *Unlimited-user System*	495	429
System V/AT *Complete System*	549	465
Runtime System *(2 Users)*	199	169
Software Development System	249	209
Text Preparation System	199	169
System V/AT *Unlimited User License*	249	209
DOSMerge286	149	125

Prices include FREE UPS surface shipping to all U.S. customers. Express services are available at the shipping carrier's standard rate.

When ordering, please specify 80286 or 80386 computer. DOSMerge 286 doesn't work on all clones, please ask before ordering.

Please refer to our main advertisement in this journal for more information and the largest advertised selection of programmer's development tools for IBM personal computers and compatibles.

Programmer's Connection
136 Sunnyside Street
Hartville, Ohio 44632

Hours: Weekdays 8:30 AM to 8:00 PM EST.

CALL TOLL FREE

USA	800-336-1166
CANADA	800-225-1166
OHIO & ALASKA (Collect)	216-877-3781
INTERNATIONAL	216-877-3781
CUSTOMER SERVICE	216-877-1110
TELEX	9102406879
EASYLINK	62806530

PROGRAMMER'S CONNECTION — QUALITY INTEGRITY Service SUPPORT PRICE

CIRCLE 98 ON READER SERVICE CARD

(continued from page 22)

9. Half Interval

The half-interval method (suggested by Jim Kajiya) assumes you have two endpoints of an arc and wish to fill in the middle with points on the circle. At each step you insert a new point between two others. Assuming a circle centered at the origin, the new point will be approximately halfway between the surrounding ones:

$$(x_m, y_m) = \left(\frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2}\right)$$

It just needs to be moved outward to lie on the circle, which involves scaling the previous expression to length 1. If the original points are at unit distance from the origin, this means dividing by $\sqrt{1 + x_1x_2 + y_1y_2}/\sqrt{2}$.

By doing this recursively, you can keep splitting until some error tolerance is met. The code is something like:

```
X1=1; Y1=0
X2=0; Y2=1
MOVE(X1,Y1)
SPLIT(X1,Y1, X2,Y2)
```

where *SPLIT(X1,Y1, X2,Y2)* is defined to be:

```
D = SQRT(2*(1+X1*X2+Y1*Y2))
XM = (X1+X2)/D
YM = (Y1+Y2)/D
IF error_tolerance_ok
  DRAW(XM,YM)
  DRAW(X1,Y2)
ELSE
  SPLIT(X1,Y1, XM,YM)
  SPLIT(XM,YM, X2,Y2)
```

The error tolerance could be just a recursion depth counter, stopping at a fixed recursion depth. This is nice because, for a given pair of initial points, the value of *D* is just a function of recursion depth and can be precomputed and placed in a table.

Pixel-Based Techniques

The other major category of algorithms involves output more directly suited to raster displays. Here the question is not where to move the "pen" next but which of the grid of pixels to light up. The preceding algorithms can, of course, be applied to pixels by generating coordinates and feeding them to a line-to-pixel drawing routine, but I won't pursue these methods. I'll just look at ways to generate the desired pixels directly. For simplicity I will assume you are drawing a 100-pixel-radius circle with pixels addressed so that (0,0) is in the center and that negative coordinates are OK. The algorithms operate in integer pixel space, assuming square pixels. Note that the following variables start with *I*, indicating that they are integers.

10. Fill Disk

Perhaps the dumbest algorithm is just to see how far each pixel is from the center and color it in if it's inside the circle:

```
FOR IY=-100 TO 100
FOR IX=-100 TO 100
    IF (IX*IX + IY*IY < 10000) SETPXL(IX,IY)
```

This, of course, fills in the entire disk instead of just drawing lines, but who's being picky? You would be correct in assuming that this might be a bit slow. Some quick speedups: calculate the value of x^2 by forward differences and calculate the allowable range of x^2 outside the x loop (forward differences probably aren't worth the trouble for this).

```
FOR IY=-100 TO 100
    IX2MAX = 10000 - IY*IY
    IX2 = 10000; IDX2 = -199; IDDX2 = 2
    FOR IX = -100 TO 100
        IF (IX2 < IX2MAX) SETPXL(IX,IY)
        IX2=IX2+IDX2; IDX2=IDX2+IDDX2
```

11. Solve for X Range Covered

The preceding algorithm still examines every pixel on the screen. You can skip some of this by explicitly solving for the range in x:

```
FOR IY= 100 TO -100 BY -1
    IXRNG = SQRT(10000 -IY*IY)
    FOR IX = -IXRNG TO IXRNG
        SETPXL(IX,IY)
```

Or just plot the endpoints instead of filling in the whole disk:

```
FOR IY= 100 TO -100 BY -1
    IX = SQRT(10000-IY*IY)
    SETPXL(-IX,IY)
    SETPXL( IX,IY)
```

This leaves unsightly gaps near the top and bottom.

12. Various Approximations to SQRT

Make a polynomial, or rational polynomial, approximation to $\sqrt{10000-y^2}$ that is good for the range $-100 \ldots +100$. Evaluate it with forward differences.

13. Driving X Away

Let's just do the upper-right quarter of the circle and follow the point (0,100). For each downward step in y, you move to the right some distance in x. Start at the x that's left over from last time and step it to the right until it hits the circle, leaving a trail of pixels behind:

```
IX=0
FOR IY= 100 TO 0 BY -1
    IX2MAX = 10000-IY*IY
    DO UNTIL (IX*IX)>IX2MAX
        SETPXL(IX,IY)
        IX=IX+1
```

Calculation of *IX2MAX* and *IX2* can be done by forward

differences:

```
IX =0
IX2=0;        IDX2=1;          IDDX2=2
IX2MAX=0; IDX2MAX=199; IDDX2MAX=-2
FOR IY = 100 TO 0 BY -1
      DO UNTIL IX2 > IX2MAX
          SETPXL(IX,IY)
          IX=IX+1
          IX2=IX2+IDX2; IDX2=IDX2+IDDX2
      I X2MAX=I X2MAX+ID X2MAX
      IDX2MAX=IDX2MAX+IDDX2MAX
```

This still has a few problems, but I won't pursue them because the next two algorithms are so much better.

14. Bresenham

The previous algorithm begins to look like Bresenham's algorithm, which is the top of the line in pixel-oriented circle algorithms. It endeavors to generate the best possible placement of pixels describing the circle with the smallest amount of (integer) code in the inner loop. It operates with two basic concepts.

First, the curve is defined by an "error" function. For my circle, this is $E=10000-x^2-y^2$. For points exactly on the circle, $E=0$; inside the circle, $E>0$; and outside the circle, $E<0$.

Second, the current point is nudged by one pixel in a direction that moves "forward" and in a direction that minimizes E. Consider just the octant of the circle from (0,100), moving to the right by 45 degrees. At each iteration, you can choose to move either to the right (R)—$x=x+1$—or diagonally (D)—$x=x+1$ and $y=y-1$.

The nice thing about this is that the value of E can be tracked incrementally. If the error at the current (x,y) is:

$$E_{cur} = 10000-x^2-y^2$$

then an R step will make:

$$\begin{aligned} E_{new} &= 10000-(x+1)^2-y^2 \\ &= E_{cur}-(2x+1) \end{aligned}$$

and a D step will make:

$$\begin{aligned} E_{new} &= 10000-(x+1)^2-(y-1)^2 \\ &= E_{cur}-(2x+1)+(2y-1) \end{aligned}$$

E_D	E_R	$E_D - E_R$
+	+	$E_D - E_R = 2y-1$, always positive
+	−	$E_D + E_R$
−	+	can never happen
−	−	$-E_D + E_R = -(2y-1)$, always negative

Table 1: *Testing the sign of* $|E_D| - |E_R|$.

Now, for the octant in question, $x \le y$, $x \ge 0$, and $y > 0$. So, an R step subtracts something from E and a D step adds something to E. The naive version of the algorithm determines which way to go by looking at the current sign of E, always striving to drive it toward its opposite sign:

```
IX=0; IY=-100
IE=0
WHILE IX<=IY
  IF (IE<0)
    IE=IE+IY+IY-1
    IY=IY-1
  IE=IE-IX-IX-1
  IX=IX+1
  SETPXL(IX,IY)
```

15. Improved Bresenham

You can do better. What you want to do at each step is actually to pick the direction that generates the smallest-size error, $|E|$. You want to look ahead at the two possible new error values:

$$\begin{aligned} E_R &= E-(2x+1) \\ E_D &= E-(2x+1)+(2y-1) \end{aligned}$$

and test the sign of $|E_D| - |E_R|$. The trick is to avoid calculating absolute values. Table 1, below, shows the possibilities.

Now comes the tricky part. You can define a "biased" error from the (+−) case:

$$G = E_D + E_R$$

and use this as the test for all three cases. This works for the following reason: In the (++) case, $|E_D| - |E_R| = 2y-1$ is positive, but so is G. In the (−−) case, $|E_D| - |E_R| = -(2y-1)$ is negative, but so is G.

G can be calculated incrementally, just like E was. The new values due to R and D steps are:

$$\begin{aligned} G_R &= G-4x-6 \\ G_D &= G-4x+4y-10 \end{aligned}$$

Further, the increments to G can be calculated incrementally. You get the idea by now . . .

```
IR=100
IX=0; IY=IR
IG=2*IR-3
IDGR=-6; IDGD=4*IR-10
WHILE IX<=IY
        IF IG<0
           IG=IG+IDGD        "go diagonally"
           IDGD=IDGD-8
           IY=IY-1
        ELSE
           IG=IG+IDGR        "go right"
           IDGD=IDGD-4
        IDGR=IDGR-4
        IX=IX+1
        SETPXL(IX,IY)
```

Whew!

Conclusion

So, why is all this interesting—aside from the pack rat joy of collecting things?

Well, you can certainly use these algorithms to optimize your circle-drawing programs if you're into circles. Each algorithm has its own little niche in the speed-accuracy-complexity trade-off space. Sometimes economy is misleading—the *SETPXL* routine often gobbles up any time you saved being clever with Bresenham's algorithm. Let's face it: unless there's something very time-critical, I usually use algorithm 1 because it's easiest to remember.

The really interesting thing about all these algorithms is the directions in which they lead you when you try to generalize them. Algorithms 2 and 7 lead to general polynomial curves. Algorithm 4 leads to iterated function theory. Algorithm 5 leads to the CORDIC method of function evaluation. Algorithm 11 has to do with rendering spheres. (I wonder what happens to algorithm 15 if you use some other simple functions of x and y for G, G_R, and G_D.) In fact, although many of these algorithms look quite similar when applied to circles, their generalizations lead to very different things. It sort of shows the underlying unity of the universe. Maybe the Greeks had something there.

Bibliography

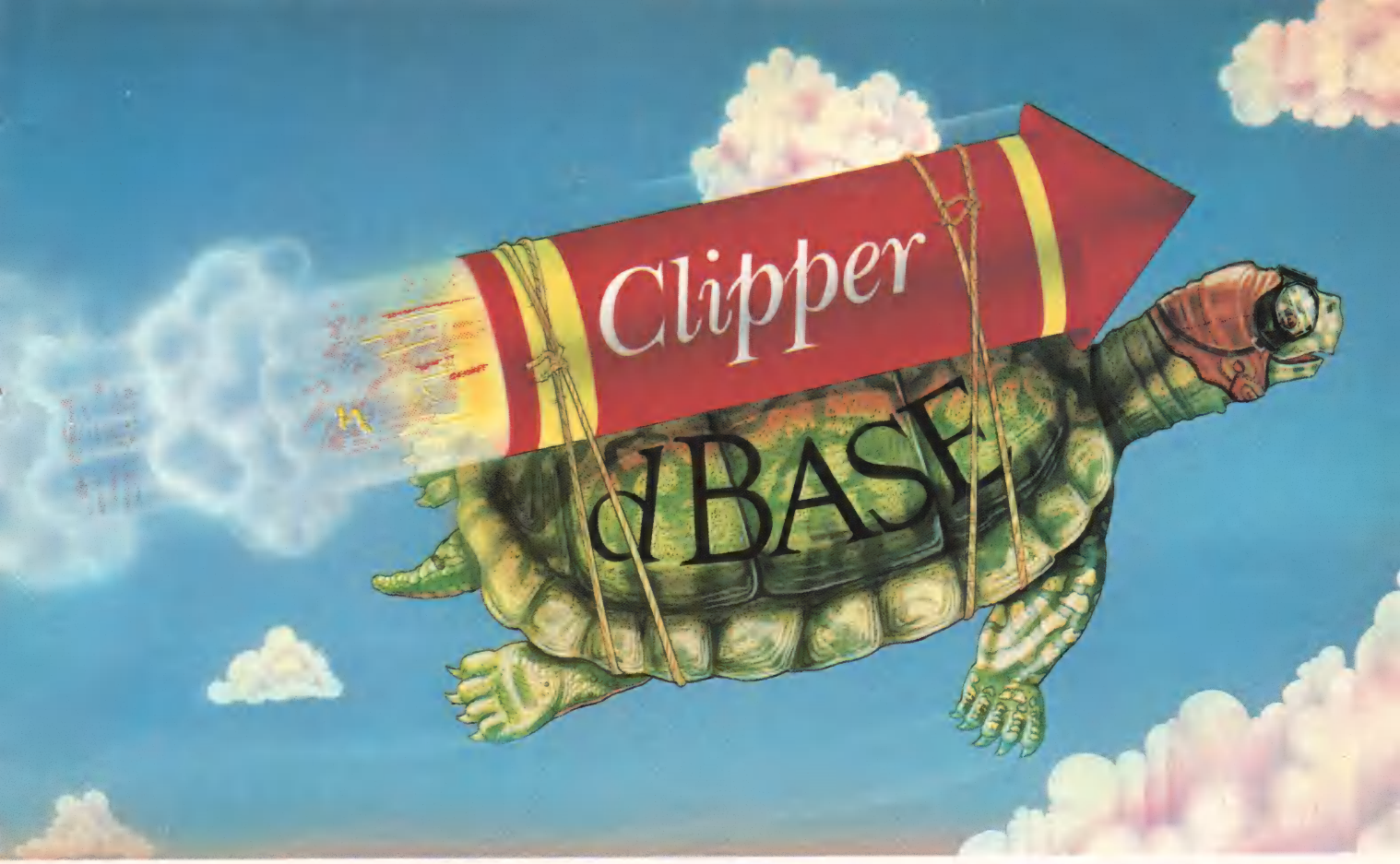
Newman, William M.; and Sproull, Robert F. *Principles of Interactive Computer Graphics*. New York: McGraw-Hill, 1979.

Turkowski, Kenneth."Anti-Aliasing Through the Use of Coordinate Transformations." *ACM Transactions on Graphics*, vol. 1, no. 3 (July 1982): 215–234. Refers to CORDIC.

For references to Runge-Kutta, see any numerical analysis text— for example, *Schaum's Outlines*.

DDJ

Vote for your favorite feature/article.
Circle Reader Service **No. 1**.



Turtle Souped

Real programmers don't use dBASE. Or do they?

We're finding that some very swift programmers are using it to write some very fast applications, and are completing their projects much more quickly.

But they cheat.

They use our Clipper™ compiler to combine dBASE™ with C and assembler.

With dBASE used like pseudo-code, they can then quickly create prototypes that actually run.

Then, with dBASE doing the high-level database functions, they use our Clipper compiler to link in C or assembly language modules from their own bag of tricks.

And they're finding that they're linking in less than they expected because Clipper compiled code runs so fast and because of Clipper's built-in enhancements.

Clipper includes easy networking that provides file and record locking the way it should be done.

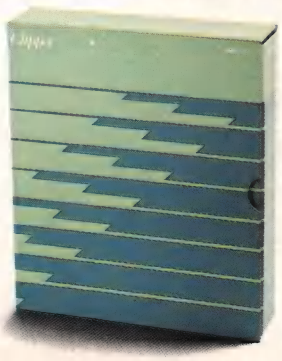
Fast screens that can be treated as memory variables and eliminate the need for direct screen writes and all that tortuous heap management code.

Box commands that make windowing a breeze. And more.

So if you'd like to use your time more productively, check us out:
Nantucket Corporation,
12555 W. Jefferson Boulevard,
Los Angeles, CA 90066.

Or if you're on deadline, call (213) 390-7923 today.

Clipper could get you out of the soup.



Nantucket®

File Comparison Algorithms

by Tom Steppe

Several popular algorithms exist for comparing two files. All of these actually look first for matches rather than differences. After the matching process has been completed, the remainders of the files that are not included in the matches are then reported as differences. (See Figure 1, page 29.)

The algorithms differ greatly in their conceptualization of the problem, however. In this article, I examine several algorithms for comparing text files—specifically, source code files—using a line as the basic unit of comparison. The ideas and algorithms I present here, however, can be extended to other types of files and other units of comparison as well. I also present a new algorithm with some interesting properties.

Determining which files are more equal than others

Evaluating The Algorithms

Any file comparison algorithm should be evaluated according to several criteria:

- Is it efficient? Time efficiency (speed) and space efficiency (memory usage) are both practical considerations. Usually they are related to the lengths of the files being compared.
- Is it robust? No algorithm is flawless. For any given file comparison algorithm, it is always possible to concoct devious situations in which its performance appears less than perfect. The algorithm should, however, be able to produce reasonable difference reports for a variety of test cases.
- Can it let differences go undetected? No algorithm should allow a file difference to go undetected.
- Can it let matches go undetected? If an algorithm can overlook matching lines, it will report these lines as differences when they are not. If the file comparison is being performed to produce a delta file, this usually is not a major problem, even though each undetected match does increase the size of the delta file unnecessarily. If the differences are to be inspected visually, however, a report of false differences can be a serious drawback.

Say, for example, that you do not have a file comparison utility and so you have to compare two files by eye. This process is certainly tedious and prone to error, especially if some of the differences are subtle. If you now use a file comparison utility that is known to report false differences, you have to inspect the output by eye and decide which reported differences are true differences. The utility has not really done the job for you, it has only made your "by eye" inspection a smaller job that is still prone to error.

- Can it detect blocks of text that have been moved? Typically, if a block of text has been moved, it simply shows up in the report of differences as a large deletion of text at one location and a large insertion of text at another. Unfortunately, no differences within the moved block are highlighted.

When a file comparison is used to create a delta file, the ability to detect moved blocks of text is probably desirable because it can lead to smaller delta files. But, when a file comparison is performed so that the differences can be inspected visually, the ability to detect moved blocks is not always as handy as it might seem to be. Trying to report the moved blocks is often difficult and can lead to complicated reports of the differences, especially when a large block of text is moved, a piece of that block is moved to another location, a piece of that piece is moved to still another location, and so on. Also, the difference report can sometimes be overburdened by uninteresting reports of small blocks (one-line and two-line blocks of text) being moved all over the place.

Only one algorithm discussed here can inherently detect moved blocks of text. The other algorithms, however, can be extended to do so, as follows. After applying the algorithm, replace each matching line in each file with a line that is guaranteed never to match. This leaves only the differences, which could contain moved blocks of text. Next, reapply the algorithm to the transformed files. Any match that is found in this pass will represent a moved block of text (see Figure 2, page 29). Continue this process iteratively until no new matches can be found. Of course, the cost of this iterative behavior is longer execution time.

These criteria help to provide a useful basis for surveying popular file comparison algorithms.

Popular Algorithms for Finding Matches

Scan Until Next Match

The "scan until next matching se-

Tom Steppe, P.O. Box 2887, Ann Arbor, MI 48106. Tom designs and develops software written exclusively in C.

quence" algorithm is probably the oldest method of file comparison. This algorithm starts at the tops of both files and matches as many lines as possible. When a difference is detected, the next M lines are scanned until at least N consecutive matching lines are found. If a sequence of N or more consecutive matching lines is found, the process begins again after the matching sequence. If such a sequence is not found, the process begins again M lines further down in the files. This process is repeated until the ends of the files are reached.

The values of M and N can be adjusted to affect the algorithm's performance. The value of M is used to control efficiency by restricting the number of lines that will be examined while searching for a sequence of matching lines. When an improper sequence of matching lines is discovered, the algorithm can be reapplied using a new value for N that is larger than the length of the improper sequence. In this way, the algorithm will overlook the undesirable sequence because it contains fewer than N matching lines, but as is always the case, the algorithm will also overlook any legitimate matching sequences that contain fewer than N lines (see Figure 3, page 30). Unfortunately, these matching lines are then reported as differences. All too often, this algorithm produces bad reports in common situations.

Although this algorithm is often highly time efficient, requires minimal memory, and frequently produces good difference reports, it does not take long to become frustrated with its shortcomings and inherent problems and begin looking for a better solution.

Longest Common Subsequence

Think of a file as representing a sequence of lines. A subsequence of those lines is defined simply as any sequence of lines that results from removing zero or more lines from the original sequence— for example, the longest subsequence of any sequence of lines is the sequence itself, with zero lines removed. Also, a sequence of zero lines would be a subsequence of any sequence because it could be created by removing all the lines from any sequence.

The "longest common subsequence" approach to file comparison takes the two files to be compared and finds the longest sequence of lines that is a subsequence of each of the files' lines—the longest common subsequence (see Figure 4, page 30). The details of the algorithm are not discussed here, but sources of such discussions are included in the bibliography. The Unix diff command is based on this algorithm.

This algorithm provides a simple, compact formalization of the file comparison problem and produces reasonable difference reports in a variety of test cases. The reports are quite acceptable whether the comparison is being used for visual inspection of the differences or for creating a delta file. In fact, among all the algorithms discussed here, it is probably safe to say that this one con-

```
file1      file2
 A          A
 B          BB
 C          C
 D          E
 E          F

Difference report:
 ######00001######################file1
 #       2   B
 #  ========================== changed to
 #       2   BB
 ######00001######################file2

 ######00001######################file1
 #       4   D
 #  ============================
 #  deleted
 ######00001######################file2

 ######00001######################file1
 #  inserted
 #  ============================
 #       5   F
 ######00001######################file2
```

Figure 1: *File comparison algorithms actually look first for line matches and then report lines that are not included in the matches as differences. The differences are usually expressed as the changes, insertions, and deletions that can be applied to one file to make it identical to the other.*

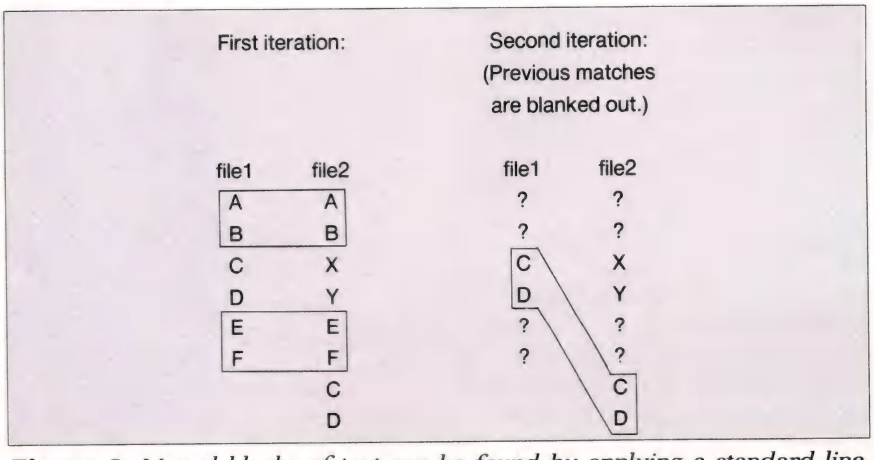


Figure 2: *Moved blocks of text can be found by applying a standard line-matching algorithm to the files and then reapplying the algorithm iteratively to the remainders of both files.*

(continued from page 29)

sistently produces the best reports when comparing files that do not involve blocks of text that have been moved.

Sometimes the quality of the reports can be overshadowed by issues of time and space efficiency. This is not always true, but situations that include a poor combination of large files and limited computer resources can lead to less than desirable performance by this algorithm. A basic implementation of the algorithm requires linear space and quadratic time. In some cases, the quadratic time can prove to be unacceptable. In summary, the "longest common subsequence" algorithm produces excellent reports, but it can be slow.

Extended Unique Line Matching

The "extended unique line matching" algorithm is based on the idea that a line that occurs once and only once in each file must be the same line. These pairs of "unique" lines determine the initial set of matched lines. (Imaginary lines at the tops and the bottoms of the files are also added to the set of matched lines.) Then, in each file, the lines adjacent to each match are examined and, if identical, are added to the set of matched lines. This process is repeated until no new matches can be found.

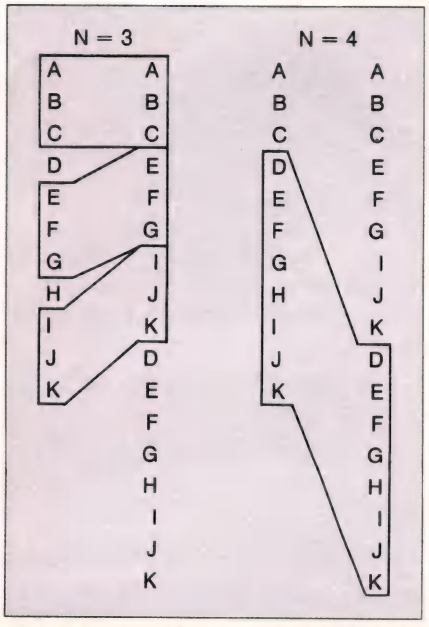


Figure 3: *The "scan until next matching sequence" algorithm often produces bad reports in common situations. When N=3, the algorithm settles for matches of three lines, never realizing that a match of eight lines is possible. When N=4, it discovers the match of eight lines but does not detect the remaining match of three lines (A, B, C).*

This algorithm has strong intuitive appeal. It is efficient, being linear in both time and space. Also, it is the only popular algorithm that inherently detects blocks of text that have been moved (even if some differences exist within the blocks). Moved blocks can be detected because the search for pairs of unique lines is in no way sequential and, therefore, can result in matches that indicate that a block of text has been moved. Note that the algorithm can find a moved block of text only if it contains a unique line match within it.

A significant problem with this algorithm is that it is prone to allowing some matches to go undetected. This occurs when matching lines are not neatly flanked by either unique line matches or the adjacent matches that have grown outward from unique line matches (see Figure 5, below).

This algorithm is fast and can frequently detect moved blocks of text, but a sacrifice is often made in the quality of the difference report. Probably its best application is in the generation of delta files when speed is the primary concern.

A New Algorithm

The "recursive longest matching sequence" algorithm uses a simple yet effective approach to the problem. This method first scans both files from beginning to end, looking for the longest sequence of consecutive matching lines. That sequence is then thought of as dividing each of the two files into an upper section and a lower section. Then, the algorithm proceeds by scanning both upper sections looking for the longest sequence of consecutive matching lines and, similarly, both lower sections for the same. These matching sequences then divide their respective sections, and the process continues recursively until no more matches can be found.

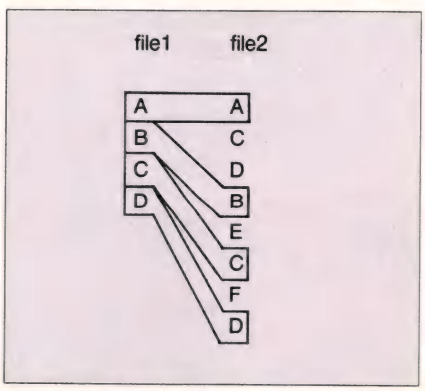


Figure 4: *The "longest common sequence" algorithm finds the longest (not necessarily consecutive) sequence of lines that is contained in both files.*

This method of file comparison is easy to understand and produces acceptable difference reports across a spectrum of test cases. It uses linear space but quadratic time. Because time efficiency can be a problem in some situations, a simple modification of the algorithm is needed. An explanation of the modification requires an understanding of the method used to locate the longest sequence of matching lines between sections of two files.

First of all, once the longest sequence is known, it can be identified by a pair of starting lines—one line from each file that specifies where the sequence begins in that file. So, when searching for the longest sequence, candidate pairs of starting lines are examined successively (in some intelligent order that starts at the beginnings of both file sections), and information is continually maintained about the length and location of the longest sequence of matching lines that has been discovered so far.

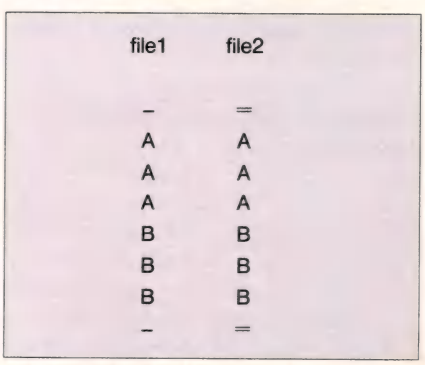


Figure 5: *The "extended unique line matching" algorithm is prone to detecting false differences. In this case, no matches are found (because there are no unique line matches) and all lines are reported as differences.*

TEACH AN OLD DOS NEW TRICKS

Protected Mode and 32-bit Performance Today

Introducing OS/286™ and OS/386™, extensions to MS-DOS 3.x that enable full use of the 80286 and 80386. Now you get direct access to all available memory, not just an archaic 640K.

OS/286 and OS/386 propel your programs beyond the limitations of DOS, without forcing you to start all over.

Moving to protected mode is simple because OS/286 and OS/386 give you *the same interface as DOS*. The hardware is still under your direct control, many 16-bit compilers already generate code suitable for OS/286 and OS/386, and existing highly tuned, machine-specific subroutines running in real mode can be efficiently called from within protected mode. Since most of your own code won't need to be rewritten, your programming investment is preserved. And because OS/286 and OS/386 work *with* DOS 3.x, they don't affect other programs, device drivers, or TSRs.

In addition to the larger address space offered by protected mode, OS/386 adds 32-bit performance to systems like the Compaq™ 386 which, until now, have been shackled to 8086 emulation.

Dhrystone Benchmarks	Microsoft C 16-bit	High C OS/386 32-bit	Scale
IBM AT 6Mhz	793	na	1.0
Compaq 386-16Mhz	2,380	5,837	7.3
HummingBoard-16Mhz	2,777	6,718	8.5
HummingBoard-20Mhz	3,571	8,470	10.7
Vax 8600 (Unix 4.3 BSD,cc)		6,423	8.1
Sun 3/160 (Sun 4.2 3.0A,cc)		3,246	4.1

OS/386 can be customized to give unmodified DOS programs up to 900k on 386 systems, regardless of how many TSR's, networks, disk caches, etc., are installed.

OS/286™ and OS/386™ Features:

- Huge address space (4GB on the 80386)
- 32-bit performance (80386)
- No rewriting of device drivers
- Compact code (under 64k)
- Support for all existing DOS calls
- New INT-21 calls for manipulating segments, invoking real-mode routines and interrupt handlers, and addressing physical memory directly
- Full interrupt vector support
- Powerful debugging: concurrent DOS environment while debugging protected mode programs
- The ability to run non-Windows programs in a window

A.I. Architects gives you a complete development toolkit:

OS/286 or OS/386 kernel and linker, Symbolic debugger and command processor

Options include:

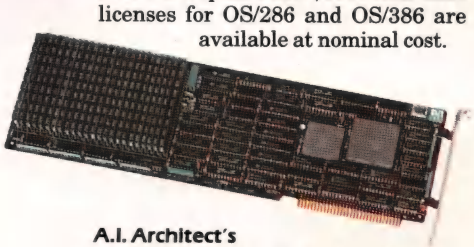
16-bit and 32-bit compilers

High C, Professional Pascal, or F77L FORTRAN

32-bit Assembler

386 HummingBoard™

The basic Developer's Kit is $495. 32-bit Compilers are $895. Run time licenses for OS/286 and OS/386 are available at nominal cost.



A.I. Architect's HummingBoard™ is a high performance 386 coprocessor for the PC-XT, AT and compatibles available with the 80387 and 2-24 Mbytes of RAM.

A.I. Architects, Inc.
One Kendall Square
Cambridge, Massachusetts 02139
(617) 577-8052

OS/286, OS/386 and HummingBoard are trademarks of A.I. Architects, Inc., Compaq Deskpro 386 is a trademark of Compaq Computer Corp., High C and Professional Pascal are trademarks of Metaware, Inc., F77L FORTRAN is a trademark of Lahey Computer Systems, Inc., Microsoft and MS-DOS are trademarks of Microsoft Corp., VAX 8600 is a trademark of Digital Equipment Corp., Sun 3/160 is a trademark of Sun Microsystems, Inc., Unix is a trademark of AT&T.

CIRCLE 265 ON READER SERVICE CARD

(continued from page 30)

When the ends of the file sections are reached, the longest sequence is known and information about the sequence is reported.

The modification to this algorithm allows the searching to stop if a sequence of N matching lines is found, realizing that it might not be the longest sequence that would be discovered if the searching were allowed to continue to the ends of the sections.

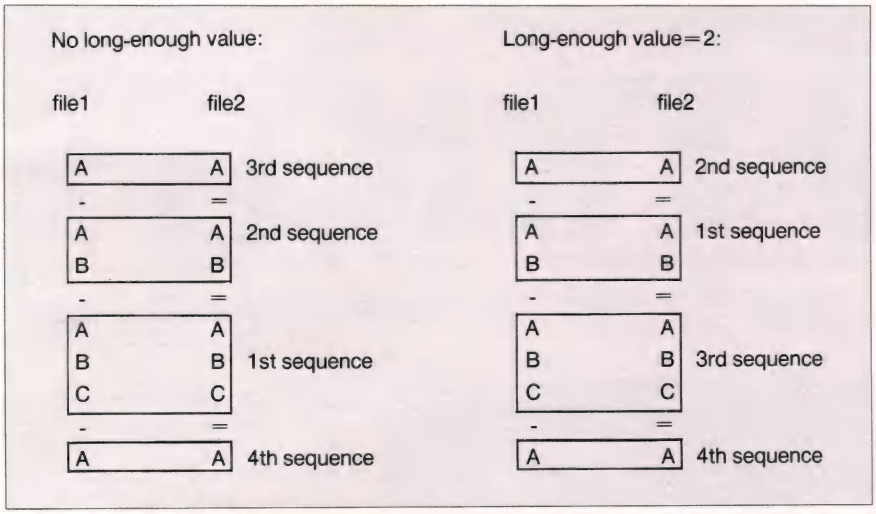


Figure 6: *With the "recursive longest matching sequence" algorithm, the use of a long-enough value often finds exactly the same sequences of matching lines although the discoveries may occur in a different order.*

This allows the searching to end prematurely (before the longest sequence has been assured) and can save considerable time. N is called the "long-enough" value. The effects of the long-enough value can be examined by choosing some test pairs of files and comparing the behavior of the algorithm when a long-enough value is used and when one is not used. Quite often, the use of a reasonable long-enough value will find exactly the same sequences of matching lines (although the discoveries may occur in a different order), thus producing an identical report of the differences but with a significant improvement in speed (see Figure 6, page 32). In fact, the use of a reasonable long-enough value allows this algorithm to perform in essentially linear time for typical cases, overcoming the previous worry of time efficiency.

The long-enough value is a parameter that you can specify. To determine a good value for your purposes, first guess at the length of the longest

Delta Files and User Reports

A file comparison utility is a versatile tool for a range of situations. It is useful to partition these situations into two distinct cases.

In the most common case, a file comparison is performed so that the differences between two versions of a text file can be inspected visually. The differences are usually expressed as the changes, insertions, and deletions that can be applied to one file to make it identical to the other file. In this case, the primary job of the comparison is to produce a concise and readable report of the differences.

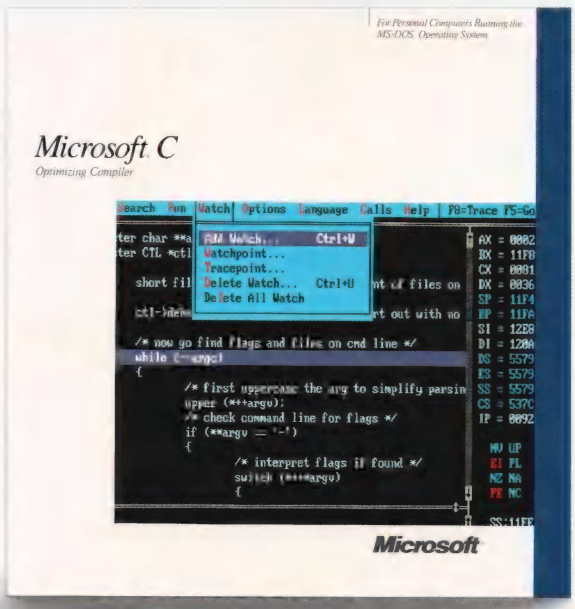
In the course of editing, a file comparison can be used in this way to highlight the differences between a previous version of a file and the current version. Valid modifications can be verified, and spurious edits can be detected. As another example, if a new version of a program is produced, a partial test of its integrity could include a file comparison of its output with the output from a previous version of the program that is known to be correct. If the two outputs compare favorably, the new program passes this integrity test. If they do not compare favorably, another file comparison can be used in the debugging process to highlight the changes between a version of a source code file that is known to work and the version that does not work.

In the second case, a file comparison is performed to generate a delta file—a file that contains a report of the differences between the two files. If the file comparison is thought of as comparing an old file with a new file, a backward delta file is designed so that it contains all the information necessary to recreate the old file, given the new file. A forward delta file is designed to be able to recreate the new file, given the old file. In either case, one of the original files can be eliminated without loss of information. If the delta file is smaller than the file it allows to be eliminated, this will result in a savings of disk space. The primary job of a file comparison in this case is to produce a compact delta file.

This use of a file comparison utility is particularly common in version control systems that maintain multiple historical versions of source code files. Only the current version of a source code file is saved, whereas a backward delta file is saved for each historical version. Any historical version can be recreated by applying the appropriate delta files to the current version of the file. The savings in disk space can be tremendous. (Alternatively, some version control systems save the first version of the file and the subsequent forward delta files.)

This usage is also common in telecommunications applications where a file at one or more remote sites has to be updated from a host. A forward delta file is created on the host by comparing the new file with the old file (a copy of the file that exists at the remote site). If the delta file is small, it is often more efficient to transmit the forward delta file and apply it to the old file than it is to transmit the new file in its entirety.

C5.0 has three features professional programmers can't live without.



Speed.

Fast Execution Speed.

	Microsoft® C 4.0	Microsoft C 5.0
Sieve (25 iterations)	5.7	3.3
Loop	11.0	0.0*
Float	19.9	0.1
Dhrystone	22.8	19.1
Pointer	14.2	7.4

- New optimizations generate the fastest code:
  - —Inline code generation. **NEW!**
  - —Loop optimizations: **NEW!**
    - —Loop invariant expression removal. **NEW!**
    - —Automatic register allocation of variables. **NEW!**
  - —Elimination of common sub expressions.
  - —Improved constant folding and value propagation.
- Fine tune your programs for even greater speed:
  - —Coding techniques for writing the fastest possible programs are included in the documentation. **NEW!**
  - —Segment Allocation Control:
    - —Group functions into the same segment to get faster NEAR calls. **NEW!**
    - —Specify which segments receive variables to yield faster NEAR references. **NEW!**
  - —Uses register variable declarations.
  - —Mix memory models using NEAR, FAR & HUGE pointers.

Benchmarks run on an IBM® Personal System/2™ *Time is negligible.

Speed.

Fast Compilation.
Fast Prototyping.

Microsoft C Version 5.0 includes QuickC,™ which lets you edit, compile, debug, and execute in an integrated environment. It's ideal for prototyping.

- In-memory compilation at over 10,000 lines/minute. **NEW!**
- Built-in editor with parentheses, bracket and brace matching.
- Use the integrated debugger to animate through your program, add watch variables and set dynamic breakpoints. **NEW!**
- MAKE file is automatically generated for you. Simply indicate the modules you want to use, then MAKE recompiles and links only those modules that have changed. **NEW!**
- Full C 5.0 compatibility:
  - —Completely source and object code compatible.
  - —Emits CodeView®-supported executables.
  - —Identical compile/link command line switches.

And speed.

Fast Debugging.

Microsoft C Version 5.0 includes Microsoft CodeView, our source-level windowing debugger that lets you debug more quickly and thoroughly than ever before.

- Debug larger programs:
  - —Debug through overlays created by the Microsoft overlay linker. **NEW!**
  - —Expanded Memory Specification (EMS) support. **NEW!**
- Fast debugging through precise control of your program execution:
  - —Access source level and symbolic debug information from your Microsoft C, FORTRAN, and Macro Assembler programs. **NEW!**
  - —View your source code and assembly simultaneously.
  - —Watch the value of variables change as you execute.
  - —Set conditional breakpoints.
  - —Animate or single step through your program.
- CodeView brings you as close as you've ever been to your hardware:
  - —Swap between your code and output screens.
  - —Watch your registers and flags change as your program executes.

Microsoft®

C 5.0 will be available soon. If you purchase Microsoft C 4.0 after June 1, 1987, we'll give you a C 5.0 upgrade. Free. For your free information packet, call:

(800) 426-9400.

0587 Part No. 098-048-615

sequence of lines you can imagine appearing more than once in a typical file. The long-enough value should be at least one larger than your guess. This will help the algorithm to avoid matching the wrong instance when a sequence of lines appears multiple times in a file. If a particular choice of long-enough value produces unsatisfactory difference reports, the algorithm can always be applied again with a larger value. When comparing C source code, I typically choose a generous value of 25, and I rarely have to rerun the comparison.

The "recursive longest matching sequence" algorithm is particularly well suited to take advantage of some common hash code technology as a means of improving time performance even more. In applications that involve repetitive string comparisons, it is often useful to calculate hash codes initially for all the strings. Then, the hash codes are compared instead of the strings themselves. The comparison of two hash code values is much quicker than is the comparison of two strings. If the hash codes are not equal, the strings cannot possibly be the same and need not be compared. If the hash codes are equal, only then must the strings be compared to prove or disprove their equality.

The performance benefits are even more dramatic when hash codes are used with the "recursive longest matching sequence" algorithm. When searching for the longest sequence of matching lines, strings do not have to be compared every time a pair of matching hash codes is found. Instead, strings only have to be compared once a sequence of matching hash codes is found that is longer than the longest sequence yet found.

The time efficiency can be improved even further if a hash code table is maintained for each file. The table should consist of an array that contains as many elements as there are possible hash code values. Each element of the array should consist of a linked list of line numbers for lines whose hash code values are equal to the array index. This table can easily be created by processing each line in the file, calculating its hash code value, and adding its line number to the proper linked list. Now, while searching for the longest sequence of matching lines by examining pairs of starting line numbers, the number of candidate pairs can be greatly reduced. For any given line in one file, only those lines in the other file that have the same hash code value (as can be easily determined from the file's hash code table) need to be considered.

A basic C implementation of the "recursive longest matching sequence" algorithm is shown in Listing One, page 54. Its simplicity, combined with a long-enough value modification and some clever use of hash codes, makes it a viable solution to the file comparison problem. It is suitable for both delta creation and visual inspection purposes.

Availability

All the source code for articles in this issue is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal*, 501 Galveston Dr., Redwood City, CA 94063, or call (415) 366-3600, ext. 216. Please specify issue number and format (MS-DOS, Macintosh, Kaypro).

You can also purchase a full-featured executable version of this algorithm from Stepping Stone Software, P.O. Box 2887, Ann Arbor, MI 48106 for $30. The available format is MS-DOS 5¼-inch DSDD.

Bibliography

Heckel, Paul. "A Technique for Isolating Differences Between Files." *Communications of the ACM*, vol. 21, no. 4 (April 1978): 264–268.

Hirschberg, D. S. "A Linear Space Algorithm for Computing Maximal Common Subsequences." *Communications of the ACM*, vol. 18, no. 6 (June 1975): 341–343.

Wagner, Robert A.; and Fischer, Michael J. "The String-to-String Correction Problem." *Journal of the Association for Computing Machinery*, vol. 21, no. 1 (January 1974): 168–173.

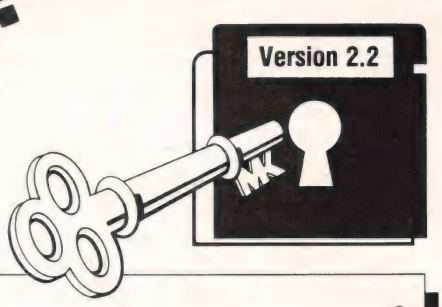
DDJ

(Listing begins on page 54.)

Vote for your favorite feature/article.
Circle Reader Service **No. 2.**

MASTER*KEY

Unlocks Everything!



turn this into this!

```
C:>DEBUG PROGRAM.COM
-D100 136
8848:0100  EB 18 49 6E 63 6F 72 72-65 63 74 20 44 4F 53 20  k.Incorrect DOS
8848:0110  76 65 72 73 69 6F 6E 0D-0A 24 50 B4 30 CD 21 86  version..$P40M!.
8848:0120  E0 3D 36 01 72 05 3D 0A-02 76 09 BA 02 01 B4 09  `=6.r.=..v.:..4.
8848:0130  CD 21 CD 20 58 EB 2F                             M!M Xk/
-Q
```

MASTER*KEY
No Other Product Comes Close!

An EXPERT may not know the solution, but always knows where to find it.

MASTER*KEY HELPS ANYONE solve those confusing and frustrating software puzzles more rapidly and easily than any other software available, at any cost! It gives you know-how within hours that may otherwise take years of experience. Create a new program from an old one. DON'T REINVENT THE WHEEL!

MASTER*KEY - Smart!
MASTER*KEY is an intelligent self-documenting MS-DOS reverse assembler. Its sophisticated procedures swiftly race through massive and baffling object code files to effortlessly discover potential trouble spots.

MASTER*KEY - Educational!
YOU DON'T NEED TO KNOW ASSEMBLY LANGUAGE! MASTER*KEY will take any program from your IBM-compatible computer and return fully-documented, easily-understood assembly language source code (Microsoft MASM 4.0 compatible).

MASTER*KEY - Easy To Use!
MASTER*KEY works both automatically from the DOS command line or interactively from menus similar to Lotus Corporation's 1-2-3 or Symphony. No need to remember any new commands or continually refer to a manual. Use it immediately!

Minimum System Requirements:
256K + 8088/8086/80186/80286/80386 PC
MS-DOS or PC-DOS 2.0 +
One 360K DSDD Floppy Drive (IBM PC Format)

MS-DOS is a trademark of Microsoft.
PC-DOS is a trademark of IBM.

```
H00100: JMP     Short H0011A                    ;00100 EB18          __
;-----------------------------------------------
        DB      "Incorrect DOS version"         ;00102 496E636F727265
        DB      0Dh                             ;00117
        DB      0Ah                             ;00118
        DB      "$"                             ;00119 24
;-----------------------------------------------
H0011A: PUSH    AX                              ;0011A 50            P
        MOV     AH,30h                          ;0011B B430          _0
        INT     21h             ;1-DOS_Ver_Number;0011D CD21         _!
        XCHG    AH,AL                           ;0011F 86E0          __
        CMP     AX,0136h                        ;00121 3D3601        =6_
        JB      H0012B          ; . . . . . .   ;00124 7205          r_
        CMP     AX,020Ah                        ;00126 3D0A02        =__
        JBE     H00134          ; . . . . . .   ;00129 7609          v_
H0012B: MOV     DX,0102h                        ;0012B BA0201        ___
        MOV     AH,09h                          ;0012E B409          __
        INT     21h             ;1-Display_String;00130 CD21         _!
        INT     20h             ;TERM_normally:20h;00132 CD20        _
;-----------------------------------------------
H00134: POP     AX                              ;00134 58            X
        JMP     Short H00166                    ;00135 EB2F          _/
;-----------------------------------------------


MASTER*KEY XREF - PROGRAM.XRF                                   Page 1

0102h                   :     121   2F5   301   320
020Ah                   :     126
03CBh                   :     12B
1-Display_String        :     130   591   610
1-DOS_Ver_Number        :     11D
H00100                  :     100
H0011A                  :     100   11A
H0012B                  :     124   12B
H00134                  :     129   134
H00166                  :     135
TERM_normally:20h       :     132
```

NOTE: The cross-reference is by memory location within the program file!

NOTE: The output is totally Microsoft MASM-compatible.

(not copy protected)

MASTER*KEY will guide you step by step to:

1. Help you learn assembly language, if you desire.
2. Discover how any program runs or why it doesn't.
3. Alter or remove unwanted object code from any program.
4. Incorporate routines from compiled programs into other assembly language, Basic, C, or Pascal programs.
5. Make software more compatible with your computer. Be certain a questionable program won't damage your system BEFORE you run it.
6. Modify software to operate with other versions of DOS.
7. Customize your COMMAND.COM or other executable program directly or by reassembling your altered MASTER*KEY source code.

Order Now! Just $79.95

Phone orders accepted on MC or VISA
Send checks to:
Sharpe Systems Corporation
2320 E Street, La Verne, CA 91750

$82.45 (includes $2.50 shipping)
$87.65 in California (includes tax & shipping)
C.O.D. orders add $2.00

(714) 596-0070

Dealer/Distributor Inquiries Welcome

Sharpe Systems Corporation
2320 E STREET, LA VERNE, CA 91750 714-596-0070

CIRCLE 176 ON READER SERVICE CARD

MASTER*KEY should not be confused with any public domain or share ware software that may have a similar name or be a similar product.

"How to protect your software by letting people copy it."

By Dick Erett, President of Software Security

Inventor and entrepreneur, Dick Erett, explains his company's view on the protection of intellectual property.

"A crucial point that even sophisticated software development companies and the trade press seem to be missing or ignoring is this:

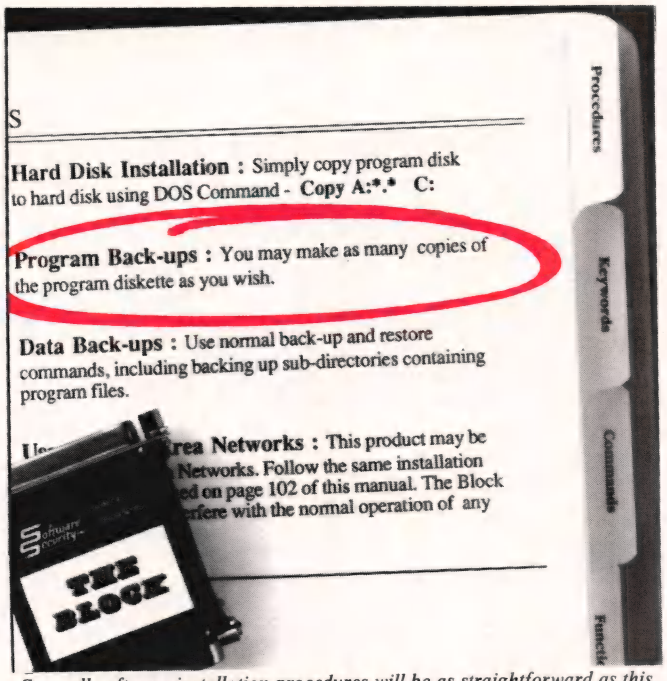
Software protection must be understood to be a distinctively different concept from that commonly referred to as copy protection.

Fundamentally, software protection involves devising a method that prevents unauthorized use of a program, without restricting a legitimate user from making any number of additional copies or preventing program operation via hard disk or LANs.

Logic dictates that magnetic media can no more protect itself from misuse than a padlock can lock itself.

Software protection must reside outside the actual storage media. The technique can then be made as tamper proof as deemed necessary. If one is clever enough, patent law can be brought to bear on the method.

Software protection is at a crossroads and the choices are clear. You can give product away to a segment of the market, or take a stand against the theft of your intellectual property.



Soon all software installation procedures will be as straightforward as this. The only difference will be whether you include the option to steal your product or not.

"...giving your software away is fine..."

We strongly believe that giving your software away is fine, if you make the decision to do so. However, if the public's sense of ethics is determining company policy, then you are no longer in control.

We have patented a device that protects your software while allowing unlimited archival copies and uninhibited use of hard disks and LANs. The name of this product is The BLOCK™.

The BLOCK is the only patented method we know of to protect your investment. It answers all the complaints of reasonable people concerning software protection.

In reality, the only people who could object are those who would like the option of stealing your company's product.

"...eliminating the rationale for copy-busting..."

Since The BLOCK allows a user to make unlimited archival copies the rationale for copy-busting programs is eliminated.

The BLOCK is fully protected by federal patent law rather than the less effective copyright statutes. The law clearly prohibits the production of work-alike devices to replace The BLOCK.

The BLOCK attaches to any communications port of virtually any microcomputer. It comes with a unique customer product number programmed into the circuit.

The BLOCK is transparent to any device attached to the port. Once it is in place users are essentially unaware of its presence. The BLOCK may be daisy-chained to provide security for more than one software package.

Each software developer devises their own procedure for accessing The BLOCK to confirm a legitimate user. If it is not present, then the program can take appropriate action.

"...possibilities... limited only by your imagination..."

The elegance of The BLOCK lies in its simplicity. Once you understand the principle of The BLOCK, hundreds of possibilities will manifest themselves, limited only by your imagination.

Your efforts, investments and intellectual property belong to you, and you have an obligation to protect them. Let us help you safeguard what's rightfully yours. Call today for our brochure, or a demo unit."

Software Security inc.

870 High Ridge Road Stamford, Connecticut 06905

203 329 8870

CIRCLE 170 ON READER SERVICE CARD

The XOR Chain Revisited

by Bennette R. Harris

Managing trees without stacks

In "The XOR Chain" (*DDJ*, June 1987), David Cortesi showed how to use the exclusive-OR (XOR) operation to compress two pointer items into one link field in a doubly linked list. This trick is just one of many that have been used for this purpose over the years. In this article I'll describe some similar tricks for doing the same thing, discuss their relative advantages and disadvantages, and then present a package of C functions that use the XOR technique to manipulate binary trees of arbitrary depth without using stacks.

The Tricks of the Trade

Cortesi's article discussed a few important properties of the XOR operation:

A XOR 0 = A (1)
A XOR A = 0 (2)
(A XOR C) XOR A = C (3)
(A XOR C) XOR C = A (4)

Properties 3 and 4 make it possible to store two pointers, A and C, in a single field as A XOR C and then recover either one of the two provided the other is known. Thus, if the addresses A, B, and C point to three nodes in sequence, and if the link field W of node B is set so that:

W = A XOR C

then W XOR A yields C, permitting travel in one direction through the list, whereas W XOR C yields A, permitting travel in the other direction.

The basic purpose of a storage technique such as this, as the preceding example illustrates, is to permit travel in two directions through a data structure while storing only a single link field. It assumes that you entered the data structure at a predefined entry point (such as a head or root node) and that an adjacent pair of nodes are referenced at any given point in time.

It is not essential that you use the XOR operation for combining the addresses—you can use almost any reversible operation. For example, provided the addresses are in a range that won't generate overflow errors, you could use the operations:

```
W = A + C
C = W - A
A = W - C
```

to move about in the data structure. This observation might be handy if you wanted to use this technique in an environment in which XOR was not available. The chief advantages of using XOR are its speed and never getting results that are out of range.

Any such technique exchanges a fairly minimal amount of computation for a reduction in the amount of storage required to hold pointers. All, however, suffer from the problem that the data stored in a link field is "unnatural"—it does not actually represent a pointer to any one node of the data structure. Instead, its contents are encoded, and the information can only be recovered using an adjacent node's pointer as a key. If a portion of the data structure becomes corrupted, the data structure as a whole becomes practically worthless because it's virtually impossible to make any sense out of links in nodes located beyond the point of corruption. Also, such link fields are hard to trace from a dump of the data structure.

You can achieve the same space savings while maintaining "natural" links by reversing the link field of the current node to point backward to the previous node as you go down the data structure and then reversing the link again as you go back up. This way, the link field remains a true pointer to nodes of the data structure. Supposing that addresses A, B, and C pointed to three nodes in sequence, the link field W of node B would initially be set so that W = C. Then, as node B was accessed traveling forward, a series of statements similar to the following would be executed:

```
NextNode = W
W = PrevNode
```

On the return back through the structure, of course, the link field of

Bennette R. Harris, 231 S. Janesville St., Whitewater, WI 53190. Dr. Harris is an assistant professor of mathematics and computer science at the University of Wisconsin-Whitewater. He teaches in the Management Computer Systems program and he also serves as a consultant both privately and through the university's Small Business Development Center.

node B would be reset using the same statements. For an example of this technique in action, see Allen Holub's discussion of nonrecursive tree traversal in C Chest, *DDJ*, July 1986.

This traversal technique exchanges the XOR computations for the overhead required to reverse the link fields. Although links are always true pointers to interesting places, a disadvantage in this case is that the data structure must be exited from completely in the reverse manner to that in which it was entered in order to restore pointers to a useful state. This is undesirable if the data structure is a large linked list, although it's perhaps acceptable in a tree. A second undesirable feature of this method is that, if a program or computer were to go down while in the middle of such an operation, the data would again be corrupted, although not irreparably so. Because the likelihood of such problems seems to be relatively high in a microcomputer environment, this is usually not acceptable. Finally, observe that the data must be accessed twice—once to read it and once to write it again with the new link. In some applications this could represent a significant amount of overhead.

Given the alternatives described here, the XOR chain stands out as one of the safest link compression techniques because the link fields are not changed in traversing the data structure.

Climbing the Tree

To keep matters simple, I will focus on binary trees in this article. Those of you who have experience with more general tree structures will be able to make the necessary adjustments.

In a binary tree, each node contains two link fields—a left link and a right link—which point to the two children of the current (parent) node. Routines for manipulating trees often make use of stacks or recursive programming techniques to maintain the back-pointers required by these routines. In many cases the necessary stack space must be set aside in advance, limiting the potential depth of the tree. With the XOR chain technique, the link fields can be coded so they can be used to travel either from parent to child or from child to parent, much as in the case of linked lists. This makes it possible to implement a binary tree without setting aside a stack to maintain back-pointers for traversing the tree.

The only reason why implementing a tree is not trivial has to do with the fact that a node has two children. Suppose you wish to visit the nodes in a tree in the fashion known as "in-order," visiting the left child, then the parent, and finally the right child. When you return from a child to its parent, you need to know whether the child was a left child or a right child. If the child was a left child, then the parent must be visited next. If the child was a right child, then the parent has already been visited, and it is time to return to the parent's parent. This is not hard to determine if the links of the parent can be matched with the node just visited, but when these links have been coded as described earlier, it is not so simple.

The chief advantages of using XOR are its speed and never getting results that are out of range.

There are at least two ways out of the difficulty. One is to include an extra bit in each node, indicating whether it is a left or right child. This extra bit is a small sacrifice for the convenience of the doubly linked pointers but still might be wasteful in a large data structure. The other way is available only if the binary tree is sorted. If the tree is sorted so that (for example) the key field of the left child is less than the key of the parent, which is in turn less than or equal to the key of the right child, then whether a node is a left child or a right child can be determined by comparing the child's key field with that of its parent. Because binary trees are frequently used to index other data structures, this criterion is often met.

Defining an Item

The structure of an item (or node) in the tree is defined as follows:

```
struct Item
{
   int key;
   unsigned llink,
      rlink;
   /* other stuff */
};
```

The key field contains the key based on which the tree is sorted. Here, I have chosen *key* to be of type *int*, but you could use other types, depending on your application. The link fields contain the XORed linking addresses. The "other stuff" is application specific.

I assume that *Items* are created and destroyed as in Cortesi's article:

```
extern struct
   Item *MakeItem( );
extern void
   DropItem(i) struct Item *i;
```

Defining a Tree

A binary tree consists of a collection of zero or more *Items* that satisfy a hierarchical relationship. If a tree is not empty, then it has a unique root item that may have zero, one, or two children. Each child item is the root of a subtree that is itself a tree. The root item of a tree is an *Item* just like any other in the tree, except that it is not the child of any other item. It is convenient not to store any data in the root item so that an empty tree consists of a root item with no children. This makes it possible to define a function to test for an empty tree:

```
int Empty(r)
struct Item *r;
{
   return( (NULL == r->llink) &&
           (NULL == r->rlink) );
}
```

I'm assuming that the tree is sorted in-order fashion—that is, the key of the left child must be less than the key of the parent and the key of the right child must be greater than or equal to the key of the parent. I'm giving the root item an artificially high key so that the data-containing items are all found in the root's left subtree. The root should be initial-

ized with:

```
root->key = MaxKey
```

where *MaxKey* is an appropriately defined value greater than any other key in the tree.

Traversing the Tree

To travel within a tree, you must enter at its root and then move up or down the tree from child to parent or parent to child. At any point within the tree, your position is described by a child-parent pair of pointers that contains the addresses of the item (child) being visited and its parent. As in Cortesi's article, these addresses are stored, together with the address of the tree's root, in a record called a *Scan*:

```
struct Scan
{
  struct Item *parent,
       *child,
    *root;
};
```

Before it can be used, a *Scan* must first be associated with a particular tree by having its root field set:

```
void Associate(s,r)
struct Scan *s;
struct Item *r;
{
  s->root = r;
}
```

Once a *Scan* is associated with a particular tree, it can be positioned at the root of the tree to begin the tree traversal:

```
void ToRoot(s)
struct Scan *s;
{
  s->parent = NULL;
  s->child = s->root;
}
```

Because the root item of a tree has no parent, you can very easily test to see if a *Scan* is at the root of the tree:

```
int AtRoot(s)
struct Scan *s;
{
  return(s->parent == NULL);
}
```

This function can be used to detect the end of a sequential walk through the tree because the root's key value is greater than all other keys in the tree and so it will be visited last.

Moving the Scan

Three fundamental actions must be described for moving around within the tree: a move from the parent to the left child, a move from the parent to the right child, and a move from a child to its parent. The first two are easy and are described by the code in Example 1, below.

The third move is more interesting (see Example 2, below). You must be able to recover the parent's parent to implement a move from the child to its parent. To do so, you must determine whether the child is a left or right child so you can XOR with the proper link field. Naturally, the *IsLeft* function needs to be modified if the necessary position information is coded differently in the tree, as would be the case for height-balanced trees that allowed nonunique keys.

Notice that the procedures for these three actions are not recursive and do not require the use of a stack for back-pointers.

Once the three fundamental actions have been described, you can write procedures that will process the tree sequentially in either direction as if it were a linked list. I'll do an in-order traversal here—that is, visit the left subtree recursively, then the root, then the right subtree. First, you need to be able to move forward and backward within the tree's list, as shown in Example 3, page 39. Both these procedures are designed to stop at the root item to make it easy to test for the ends of the tree's list.

To start sequential processing, you first have to locate the head or tail of the list:

```
void ToHead(s)
struct Scan *s;
{
  ToRoot(s);
```

```
void GoLeft(s)
struct Scan *s;
{
  struct Item *i;

  i = s->parent↑s->child->llink;
  s->parent = s->child;
  s->child = i;
}

void GoRight(s)
struct Scan *s;
{
  struct Item *i;

  i = s->parent↑s->child->rlink;
  s->parent = s->child;
  s->child = i;
}
```

Example 1: *Moving from the parent to its children*

```
int IsLeft(s)
struct Scan *s;
{
  return(s->child->key < s->parent->key);
}

void GoParent(s)
struct Scan *s;
{
  struct Item *i;
  if (IsLeft(s))
    i = s->parent->llink↑s->child;
  else
    i = s->parent->rlink↑s->child;

  s->child = s->parent;
  s->parent = i;
}
```

Example 2: *Moving from a child to its parent*

```
  while (s->child->llink != s->parent)
    GoLeft(s);
}

void ToTail(s)
struct Scan *s;
{
  ToRoot(s);
  GoBak(s);
}
```

Inserting an Item

Unlike the case for a linked list, new items cannot be inserted at arbitrary places within the tree; instead, new items must be added to the tree in such a way that the tree remains sorted.

In the insertion procedure shown in Example 4, below, the scan is passed to identify the tree in which the item is to be inserted. The tree is traversed (not sequentially!) until the proper insertion point for the item is found. The item is then inserted as a leaf of the tree, with no children of its own. First, the appropriate link field of the item's parent is set to point to the new item (using XOR), and then the item's links are set to point back to its parent. Finally, the scan is returned with the new item as the current active child.

Deleting an Item

Deleting an item from a binary tree is much more complicated because the resulting items must still maintain the binary structure and sorted arrangement of the tree before the item was deleted.

In my implementation, if the item has only one child, then I delete it by pointing its parent to the nonempty subtree, and vice versa. If the item has two children, I delete the item by removing it and replacing it with the greatest item in the left subtree. This item will have at most one child (a left child), and so the item can be repositioned with a minimum of manipulation of the link fields. Thus, the procedure used here must handle four cases:

1. The item to be deleted has no children (an item has no children if its link fields are equal to each other because both would point back to the item's parent).
2. The item has a right child but no left child (an item has no left child if the left link equals the pointer to the item's parent).
3. The item has a left child but no right child.
4. The item has both a left and a right child. There are two possibilities:

a. If the left child has no right child of its own, then the left child replaces the item and picks up the item's right subtree.

b. If the left child has a right child, you travel down through right children as far as you can go to locate the greatest item less than the item to be deleted.

These cases are more difficult than usual, even for a tree delete routine, because not only must parent pointers be adjusted to point to their new children but also child pointers must be adjusted to point back to their new parents. This means testing to make sure the children exist before accessing their link fields.

When the deletion procedure (see Listing One, page 66) is called, the node to be deleted is pointed to by *s->child*. The scan is set back to the parent (by a call to *GoParent*) before the routine returns so that no scan is ever returned with a *NULL* child (because the child item of the scan is the item being visited). In fact, *GoParent*

```
void GoFwd(s)
struct Scan *s;
{
  if (s->child->rlink != s->parent)
  {
    GoRight(s);
    while (s->child->llink != s->parent)
      GoLeft(s);
  }
  else
  {
    while ( !AtRoot(s) && !IsLeft(s))
      GoParent(s);
    if ( !AtRoot(s))
      GoParent(s);
  }
}

void GoBak(s)
struct Scan *s;
{
  if (s->child->llink != s->parent)
  {
    GoLeft(s);
    while (s->child->rlink != s->parent)
      GoRight(s);
  }
  else
  {
    while ( !AtRoot(s) && IsLeft(s))
      GoParent(s);
    if ( !AtRoot(s))
      GoParent(s);
  }
}
```

Example 3: *Moving backward and forward within the tree's list*

```
void Insert(i,s)
struct Item *i;
struct Scan *s;
{
  ToRoot(s);

  while (s->child != NULL)
    if (i->key < s->child->key)
      GoLeft(s);
    else
      GoRight(s);

  if (i->key < s->parent->key)
    s->parent->llink ↑= i;
  else
    s->parent->rlink ↑= i;

  i->llink = s->parent;
  i->rlink = s->parent;
  s->child = i;
}
```

Example 4: *The insertion procedure*

(continued from page 39)

is called early in the procedure, before any pointers are rearranged.

Conclusion

As with any arbitrary binary tree system, it is possible for the tree to become quite unbalanced after several insertions or deletions, degrading the performance of the system. A good question is whether the same idea could be woven into an AVL height-balanced tree system such as that discussed by Allen Holub (*DDJ*, August 1986, page 20). The answer is no, not in the form presented here. My routines rely heavily on the fact that the key of any item is strictly greater than the key of its left child. This cannot be guaranteed in an AVL tree if nonunique keys are allowed. In that case, it would be necessary to use the alternative approach I mentioned earlier: code a bit in each item, marking it as either a left child or a right child. With this modification, height balancing is possible.

Another interesting implementation would involve B-trees and other generalized tree structures. Such implementations are possible with unique keys, or with a few code bits added to each tree item, and can be coded in much the same way as the examples shown here.

Finally, I should say a few words about another nonrecursive traversal method—namely, threaded trees. The main problem with this technique is maintaining the threads as items are added to and deleted from the tree. By their very nature, threaded links are links to items that usually are not adjacent to the item being added or deleted, and so there are very few easy cases in the add and delete algorithms. I will leave it to you to determine which has the more substantial overhead.

A wise instructor once said, "Ninety-nine percent of all programs using recursion could be written more effectively without it." The XOR chain technique makes it possible to apply this maxim to tree structures as well. With a little thought and creativity, you should be able to customize these ideas to fit your needs.

Availability

All the source code for articles in this issue is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal*, 501 Galveston Dr., Redwood City, CA 94063 or call (415) 366-3600, ext. 215. Please specify the issue number and format (MS-DOS, Macintosh, Kaypro).

Bibliography

Holub, Allen. "C Chest: Nonrecursive Tree Traversal." *DDJ* 117 (July 1986): 18–20. There's a bug fix in *DDJ* 123 (January 1987): 103.

Holub, Allan. "C Chest." *DDJ* 118 (August 1986): 20–29.

Knuth, Donald E. *The Art of Computer Programming, Volume 3: Sorting and Searching*. Reading, Mass.: Addison-Wesley, 1973.

Singh, Bhagat; and Naps, Thomas L. *Introduction to Data Structures*. St. Paul, Minn.: West Publishing Co., 1985.

DDJ

(Listing begins on page 62.)

Vote for your favorite feature/article.
Circle Reader Service **No. 3.**

THE ADA® WORLD HAS CHANGED.

VALIDATED

Meridian's AdaVantage™ v2.0 compiler is a complete implementation of the Ada® language that has been validated on the IBM PC/XT, IBM PC/AT, and the Zenith Z-248. Most of the representation clauses and implementation-dependent features are also available including

- pragma pack
- size specification
- task storage size
- fixed point small
- record representation clauses
- address clauses
- package system
- representation attributes
- pragma interface
- unchecked storage deallocation
- unchecked type conversions

The compiler includes a complete set of utilities for managing the Ada program library and all of the standard packages including text_io, system, and calendar.

BEST PRICE PERFORMANCE

The Meridian AdaVantage compiler demonstrates the best price/performance of any PC Ada compiler. The compiler sells for $795 in single quantities and it compiles about 1000 lines per minute on an IBM PC/AT.

In addition to the production compiler, two other versions are available for general training and student use. The AdaTraining™ compiler sells for $395 and is intended for corporate and university educational environments. The AdaStarter™ compiler, priced at $129, incorporates all of the features of the AdaVantage production compiler, with certain limitations on the number of library units and the number of lines per compilation unit allowed. The full price of the AdaStarter compiler can be applied to a later purchase of the AdaVantage production compiler.

ADDITIONAL PRODUCTS

Two optional packages, priced at $50 each, that provide DOS environment support and miscellaneous utility routines are currently available. A source-level debugger and Ada editor will be available this Fall.

CONFIGURATION

The compilers all run in a standard PC configuration with 640K of memory, a hard disk, and DOS v2.1 or higher.

To order today, or get more information, call toll-free 1-800-221-2522.

MERIDIAN
SOFTWARE SYSTEMS, INC.

23141 Verdugo Drive, Suite 105
Laguna Hills, CA 92653
800/221-2522 (outside Calif.)
714/380-9800 (inside Calif.)
Telex: 650-268-0547 MCI

CIRCLE 397 ON READER SERVICE CARD

VEDIT PLUS

#1 PROGRAMMABLE EDITOR

Call 1-800-45-VEDIT for FREE Fully Functional Demo Disk

Stunning speed. Unmatched performance. Total flexibility. Simple and intuitive operation. The newest VEDIT PLUS easily satisfies the most demanding computer professional.

Try a Dazzling Demo Yourself.

The free demo disk is fully functional—you can try all features yourself. Best, the demo includes a dazzling menu-driven tutorial—you experiment in one window while another gives instructions.

The powerful "macro" programming language helps you eliminate repetitive editing tasks. The impressive demo/tutorial is written entirely as a "macro"—it shows that no other editor's "macro" language even comes close. And VEDIT PLUS is only 40K in size.

Go ahead. Call for your free demo today, You'll see why VEDIT PLUS has been the #1 choice of programmers, writers and engineers since 1980.

Only VEDIT PLUS is this Flexible.

The installation lets you pick from closely emulating the keyboard layout of Word Perfect, WordStar and others. Or you can easily create your own layout and even your own editing functions. Supports any screen size—you pick screen colors and attributes.

Supports the IBM PC, XT, AT and PS/2. Also supports MultiLink, PC-MOS/386, Concurrent DOS and most networks. Also available for MS-DOS, FlexOS (protected mode), CP/M-86 and CP/M. (Yes, we support windows on most CRT terminals, including CRTs connected to an IBM PC.) Order direct or from your dealer. $185.

Special: VEDIT (single file, no windows) for CP/M—$49.

- **Fully Network Compatible**
- **Call for XENIX-286 version**
- **30 Day Money-back guarantee**

Compare Features and Speed

	BRIEF	Norton Editor	PMATE	VEDIT PLUS
'Off the cuff' macros	No	No	Yes	Yes
Built-in macros	Yes	No	Yes	Yes
Keystroke macros	Only 1	No	No	Unlimited
Multiple file editing	20+	2	No	20+
Windows	20+	2	No	20+
Macro execution window	No	No	No	Yes
Pop-up menus	No	No	No	Yes
Execute DOS commands	Yes	Yes	Yes	Yes
Automatic processing of Compiler errors	Yes	No	No	Yes
"Cut and paste" buffers	1	1	1	36
Undo line changes	Yes	No	No	Yes
Paragraph justification	No	No	No	Yes
Convert to/from WordStar	No	No	No	Yes
On-line calculator	No	No	No	Yes
Configurable Keyboard	Hard	No	Hard	Easy
43 line EGA support	Yes	No	No	Yes
Manual size/index	250/No	42/no	469/Yes	380/Yes
Benchmarks in 120K File:				
2000 replacements	1:15 min	34 sec	1:07 min	6 sec
Pattern matching search	20 sec	Cannot	Cannot	2 sec
Pattern matching replace	2:40 min	Cannot	Cannot	11 sec

*Also available for TI Professional, Tandy 2000, DEC Rainbow, Wyse WY700 and others.
*Demo disk is fully functional, but does not readily write large files.

CompuView

1955 Pauline Blvd., Ann Arbor, MI 48103
(313) 996-1299, TELEX 701821

WINDOWS FOR DATA™

The first choice of professional C programmers

"Windows for Data is the best programming tool I've ever used.
It's the most flexible I've seen.
Whenever I've wanted to do something,
I've been able to find a way."

Steven Weiss,
Stratford Systems

Professionals choose our tools because they are designed, crafted, and supported for professionals. Here at Vermont Creative Software, we understand that performance and pleasure in programming derive from more than a long list of functions. **Windows for Data** provides:

PROFESSIONAL FLEXIBILITY: Our customers repeatedly tell us how they've used WFD in ways we never imagined - but which we anticipated by designing WFD for unprecedented adaptability. Virtually every capability and feature can be modified to meet special needs. You will be amazed at what you can do with WFD.

PROFESSIONAL PERFORMANCE: Screen output is crisp and fast. Windows, menus, and data-entry forms snap up and down from the screen. WFD is built upon and includes **Windows for C, the windowing system rated #1 in speed and overall quality** in PC Tech Journal (William Hunt, July 1985).

PROFESSIONAL RELIABILITY: An unreliable tool is worse than no tool at all. VCS products are known in the industry for their exceptional reliability. Ask anyone who owns one.

PROFESSIONAL DOCUMENTATION: Over 600 pages of documentation provide step-by-step explanations for each major application, a reference page for each function, listings of functions alphabetically and by usage, and a fully cross-referenced index. Extensive tutorials and demonstration programs assist learning.

PROFESSIONAL TECHNICAL SUPPORT: The same expert programmers that develop our products provide prompt, knowledgeable technical support.

PROFESSIONAL PORTABILITY: High-performance versions of VCS products are available for XENIX, UNIX, and VMS, as well as DOS. No royalties on end-user applications.

OUR CHALLENGE AND GUARANTEE

If you have an application where no other tool can do the job, try **Windows for Data.** If it doesn't help you solve your problem, RETURN FOR A FULL REFUND. YOU MUST BE SATISFIED.

Ask for **FREE DEMO DISKETTE**

Vermont Creative Software

21 Elm Ave.
Richford, VT 05476
Telex: 510-601-4160 VCSOFT

Tel.: 802-848-7731

Prices: PCDOS* $395; XENIX, VMS, UNIX
*PCDOS specify C compiler.

WINDOWS FOR DATA

for DOS, UNIX, VMS ...

The complete windowing data entry, menu, and help system that does the hard job others can't — we **guarantee** it!

Pop-up data entry windows; field types for all C data types, plus decimals, dates, and times; auto conversion to and from strings for all field types; system and user supplied validation functions; range checking; required, must-fill, and protected fields; free-form movement; multiple-choice field entry; scrollable sub-forms. Branch and nest windows, forms, and menus.

Complete context-sentitive help system with pop-up windows and scrollable text.

Pop-up, pull-down, scrollable, and Lotus-style menus.

NEW FOR DEBUGGING: Exclusive **VCS Error Traceback System** automatically identifies the location and cause of program errors. Eliminates the need to code error checks on all function calls! **VCS Memory Integrity Checking** helps catch those hard-to-detect, memory-corruption errors.

NEW FOR ERROR HANDLING: Install your own error handler to be called whenever a function detects an error.

NEW FORM LAYOUT UTILITY simplifies form design.

CIRCLE 157 ON READER SERVICE CARD

Writing MS-DOS Device Drivers in C

by Andy Klein

An alternative to assembly language

The MS-DOS/PC-DOS installable device driver facility has been available since Version 2.0 and allows you to add extra devices to your system without making any modifications to DOS itself. You specify devices to be added in an ASCII file called CONFIG.SYS using the command *device=xxxxxxxx.yyy*. Installable device drivers allow hardware independence as hardware-specific code is isolated in the driver.

This article shows how to write the functions that implement a device in C. To do this, I discuss the format of a device driver and how to create a driver in this format using (mostly) object code produced by a C compiler. Some of the material is compiler specific and pertains to Aztec C, but it should be portable to other C compilers (even to other high-level languages). The example device driver I present—prndrv.asm in Listing One, page 68—is a simple one that implements a parallel printer and replaces the standard PRN device. I've chosen a simple device deliberately so that I can concentrate on how to construct drivers in C instead of on the details of a specific device and at the same time provide a real, working device driver that you can modify and experiment with on your PC.

Request Header

DOS communicates with its device drivers through a packet called a request header, which is a formatted block of system memory that contains the command code that the driver is to perform, a status word that the driver uses to report the success or failure of the operation back to DOS, and the length of the packet. The header is followed by data needed for the operation, which varies depending on the operation to be performed. For my driver, this variable area can be considered fixed, containing the segment, offset, and number of bytes to transfer. *Req_hdr* is a C structure that is *typedefed* in rh.h (Listing Two, page 71) and is used by the driver implementation functions to access the request header data. An important task you need to accomplish is to make the request header addressable by C compiled functions (discussed later).

When DOS calls a driver to perform a task, it does so in two separate stages. The first stage is referred to as the device strategy and occurs when the device is passed a long (segment:offset) pointer to the request header in the *es:bx* register pair. The device does not perform the request at this stage—it just saves the pointer to the request header. The next stage is called the device interrupt. It receives no parameters; instead, the interrupt function retrieves the previously saved pointer to the request header and then performs the operation indicated by the command code in the header. It is the interrupt function that actually does the work of the device, and it also has the task of setting things up so that you can call C compiled functions. DOS always calls the strategy function, then immediately calls the interrupt function. Following this pattern, device drivers have two entry points, called the strategy and interrupt routines. These entry points are implemented in prndrv.asm as labels *dev_strategy* and *dev_interrupt*.

Format of a Driver

DOS device drivers must be in a specific format in order to be incorporated into MS-DOS. This format differs from that of a .COM or .EXE file. Drivers must start with a device header that starts at offset 0h from the start of the file, and all the logical segments of the driver must be in one physical segment, such as a .COM file, because the driver is simply loaded into memory by DOS—it will not get any of the segment fix-ups that an .EXE file receives when loaded. The largest hurdle you'll face when writing your first device driver is to get it to load at boot time without hanging up the computer. Once you get past this point, the rest is cake.

Device Header

The device header's purpose is to provide DOS with the attributes of the device, the offsets of the strategy and interrupt entry points into the driver, and the name of the device or the number of units it controls. The fields of the header are:

- pointer to next header (4 bytes)
- attribute (2 bytes)
- pointer to strategy routine (2 bytes)
- pointer to interrupt routine (2 bytes)

Andy Klein, 3801 N. 16th St. #222, Phoenix, AZ 85106. Andy's company, Micro Quantitative Sciences, develops software interfaces for the IBM PC and peripheral devices.

• name or number of units (8 bytes)

The pointer to next header is a 4-byte field that DOS uses to store a pointer to the next driver in the chain of all drivers. These four bytes should be filled with 0xff in prndrv.asm to indicate to DOS that there is only one driver in this file.

The attribute field is bit-mapped as shown in Table 1, page below. My example driver is a character device and has bit 15 set; the others are all 0s.

The name or number of units depends on the driver type. If the driver is a character device, then this field is an eight-character name, left-justified and padded with spaces. The name assigned here will be the name DOS uses for the device. When character devices are loaded at boot time, installable devices are linked into the chain of all drivers before the default or built-in character drivers. When DOS searches this linked list for a named driver, it stops on the first match, so if you use the name of a default driver as the name of your driver, you replace the default driver with your own. This is exactly what the example driver does, replacing the default PRN device with my PRN device.

Note that there is one exception to this rule—the NUL device cannot be replaced because it is the starting point of the linked list of devices. Block devices do not have names; instead, they are assigned drive letters by DOS in the order in which they are loaded. Unlike character device drivers, the default block drivers are loaded first and cannot be replaced. A block driver can control multiple subunits, and the number of units controlled is entered in this field for block devices with the last 7 bytes of the field filled with spaces. The number of subunits can be overridden by the initialization function for block devices. An example of the use of subunits is a fixed disk controller that can have two fixed disk drives attached.

Device Driver Start-Up Code

In order to write the functions that implement a device in C, a few special things have to be done. First, before any C compiled functions are called, the environment (segment registers and stack pointer) must be set up to correspond with what the compiler-generated code assumes. Normally this setup is done by the start-up code that is provided with the compiler, which for Aztec C using the small model is in the Aztec function sbegin.asm. All Aztec C compiled functions make an external reference to a function called *$begin*, causing the linker to drag it in from the standard library c.lib. *$begin* in turn makes an external reference to *Croot*, which in turn references *main*. In the normal case, *$begin* sets up the environment and calls *Croot*, which then calls *main*. Other compilers follow the same pattern.

Installable character devices are linked into the chain before the default drivers.

Much of what is done in the compiler start-up functions does not have to be replicated in the driver start-up code, including parsing *argv( )*, allocating a heap, getting access to the DOS environment variables, and other initializations that are required for some library functions. There is a penalty exacted for this approach, and it is that your C functions cannot call library functions that require some setup that you have not done—notably, *malloc( )*, *free( )*, and all those functions pertaining to I/O are off limits. Also, variables declared outside functions must be initialized to some value. An essential part of the compiler's start-up that must be incorporated into the device driver start-up code is establishing a stack frame.

In the example driver, the start-up tasks are accomplished by replacing the normal compiler start-up code with the device interrupt function. This function takes care of setting up the segment register to what Aztec C expects. There is no *main( )* because the entry point is the label *dev_interrupt* in prndrv.asm. It is necessary to provide a function called *$begin* to prevent the linker from reporting an error, so prndrv.asm has a function of this name but it is never called. Code generated by earlier versions of Aztec C also has a reference to a function called *$cswt*, which is also in prndrv.asm. Users of other compilers will have to replace these functions with whatever their compiler demands. If the source for the start-up function for your compiler is available, it is fairly straightforward. (Compiler publishers that don't make this source available significantly limit the applicability of their products for serious system development.) You can discover much of what your compiler does by compiling a few functions and getting the compiler to output the assembly-language result of the compilation.

Device Strategy Function

The strategy function is called with a long pointer to a request header in the *es:bx* register pair. The strategy function does not perform the re-

bit	15	1=character device,	0=block device
bit	14	1=supports *ioctl*,	0=no *ioctl*
bit	13	1=non-IBM format,	0=IBM format (block device)
		1=supports output until busy ,	0=doesn't (char dev)
bit	12	0 (reserved by Microsoft)	
bit	11	1=supports removable media	(block device only)
bits	10	–5 reserved, should be 0	
bit	3	1=clock device,	0=not clock device
bit	2	1=NUL device,	0=not NUL device
bit	1	1=*stdout* device,	0=not *stdout* device
bit	0	1=*stdin* device,	0=not *stdin* device

Table 1: *Bit mapping of the attribute field*

"The Ada programming language shall be the single, common, high order programming language for..

"...all computers that are integral to, physically a part of, dedicated to, or essential in real time to a performance of the mission of weapon systems... used for specialized training, diagnostic testing and maintenance, simulation, or calibration of weapon systems... used for research and development of weapon systems...Use of validated compilers is required...this directive is effective immediately."
—DoD Directive 3405.2, 3/30/87.

"...Defense computer resources used in intelligence systems, for the command and control of military forces...all major software upgrades...all other applications (some exceptions) in keeping with the long range goal of establishing Ada as the primary DoD higher order language...waivers to the policy...shall be strictly controlled and closely reviewed... this directive is effective immediately."
—DoD Directive 3405.1, 4/2/87.

Ada NOW

Special Introductory Offer For Apollo, Altos, H-P, and Sun Users

Ada Now

New Alsys Toolset For 68000 Ada Builds Unique Project Environment

Organizations serious about the 680X0 architecture, and serious about working with the government, want a lot more than just validated Ada compilers. They want quality solutions; production quality compilers and quality programming tools.

Just what Alsys offers. Alsys' new 68000 Ada Developer's Toolset includes:

- ***AdaPROBE**, a unique source-level symbolic debugger and program viewer;*
- ***Cross-Referencer**, an inter-unit cross-referencing utility;*
- ***Reformatter**, a pretty printing tool for reformatting source files to selectable conventions; and*
- ***AdaMAKE**, an automatic recompilation facility.*

*Consider, too, all those special Ada "manager tools" that are part of the Alsys Version 3 compilation system: the **Family Manager**, the **Unit Manager**, and the **Library Manager**.*

*Together, they implement the new Alsys **Multi-Library Environment** that allows teams of programmers to share thousands of logically organized compilation units.*

Alsys 68000 compilers are in a class by themselves; highest code quality, maturity, reliability, robustness, superior optimization technology, unexcelled error messages...And now, with the new development tools, they are at the core of an Ada project environment unique in the industry.

Here is our special INTRODUCTORY OFFER. Between now and October 31, 1987, order any of our 68000 Ada compilers and we will include the complete Toolset FREE. AdaPROBE, Cross-Referencer, Reformatter, AdaMAKE.

Ada is Now. Alsys solutions are now. Call or write.

alsys

In the US: *Alsys Inc., 1432 Main St., Waltham, MA 02154 Tel: (617) 890-0030*
In the UK: *Alsys Ltd., Partridge House, Newtown Rd., Henley-on-Thames, Oxon RG9 1EN Tel: 44 (491) 579090*
In the rest of the world: *Alsys SA, 29 Avenue de Versailles, 78170 La Celle St. Cloud, France Tel: 33 (1) 3918.12.44*

The Many Facets of **Quality**

_____ YES, I'm interested in your Introductory Offer. Send more information on the Toolset and your 68000 compilers.

_____ Send me your free brochure on The Many Facets of Quality.

Name_____

Company_____

Address_____

City_____ State_____ Zip_____

Phone_____

Alsys, Inc. • 1432 Main Street • Waltham, MA 02154 DDJ 9/87

CIRCLE 273 ON READER SERVICE CARD

*Ada is a registered trademark of the U.S. Government (AJPO).

(continued from page 45)

quested function; it just saves the pointer to the request header and then returns. The request header pointer segment is stored in the code segment variable *req_hdr_seg*, and the offset is stored in the code segment variable *req_hdr_off* (see the listing).

Device Interrupt Function

The device interrupt function is where the command stored in the request header is carried out. Although this function is called the interrupt function, DOS invokes it as a far call as opposed to a processor interrupt. This function has seven crucial tasks to perform, some of which replace the C compiler's start-up code. The listing of prndrv.asm has the sections marked as *;STEP 0*, *;STEP 1*, and so on, where:

• *STEP 0* saves the machine state. On entry, the interrupt function saves the machine state by pushing all the registers onto DOS' stack, which has enough space to have the registers pushed onto it. Then the DOS stack frame—the *ss* and *sp* registers—is saved in the code segment variables *caller_ss* and *caller_sp*.

• *STEP 1* gets the driver's segment. The first step in establishing addressability is to get the driver's segment address and hold it in the *ax* register. Remember that the driver's segments are all in one physical segment, so when the driver is called, the value of the *cs* register is the segment value for all segments.

• *STEP 2* makes the request header addressable. The long pointer to the request header was previously saved in the strategy function. Now the interrupt function uses this information to copy it into the private request header.

• *STEP 3* finishes establishing addressability. The interrupt function moves the segment address of the driver from *ax* into the *ds*, *es*, and *ss* registers; gets the offset of the private stack and puts it into *sp* to establish the stack frame; and finally, sets the *bp* register equal to the *sp* register as this condition is needed to call C functions that rely heavily on *bp*.

• *STEP 4* calls the first C function. Now the interrupt function is ready to call the first C function, which will in turn call others to perform the requested task. It calls a function named *driver_functions(cmd_code)* and passes the command code from the request header as a parameter. Calling a C function from assembler requires pushing the parameters onto the stack, calling the function, and removing the parameters from the stack.

• *STEP 5* updates DOS' request header. The task DOS wanted the driver to perform has finished. The status of the operation is stored in the addressable copy of the request header along with any function-specific return info. In order for DOS to get this info, the request header must be copied back to the request header that DOS gave a pointer to way back in the strategy function.

• *STEP 6* cleans up. To clean up, the original DOS stack frame that was saved in *caller_ss* and *caller_sp* must be restored, then the registers pushed onto DOS' stack must be popped. Finally, the interrupt function returns, and its tasks are finished.

Clarify and document your source listing and get an "organization chart" of your program's structure

with two NEW utilities from Aldebaran Laboratories, for C, BASIC, Pascal, dBASE,® FORTRAN and Modula-2 programmers.

Now works with FORTRAN

"Occasionally, a utility comes along that makes a programmer's life much easier. SOURCE PRINT is such a program. It contributes to the programmer's job by organizing code into a legible format and by helping to organize the documentation and debugging process."

— PC Magazine Sept. 16, 1986

Source Print and Tree Diagrammer both have easy-to-use menus with point-and-shoot file selection, and let you search for files containing a given string. For IBM PC and compatibles with 256K.

Join thousands of programmers who are working more efficiently using Source Print and Tree Diagrammer. Order these indispensable tools today. We ship immediately, and there's no risk with our 60-day money-back guarantee. **Order both and save. Only $155.00.**

800-257-5773 Dept. 58
In California:
800-257-5774 Dept. 58

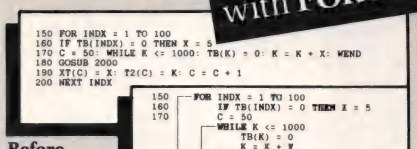
MasterCard, VISA, American Express, COD. Add $5 for shipping/handling.

or see your local dealer!

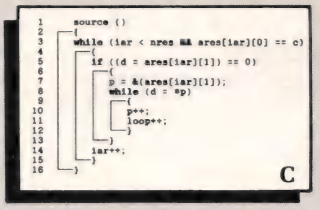
Source Print and Tree Diagrammer are trademarks of Aldebaran Labs. dBASE is a trademark of Ashton Tate. Prices subject to change without notice.

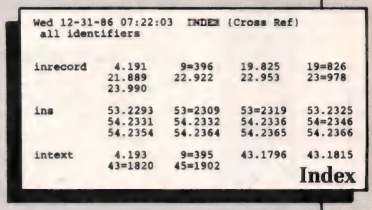
Source Print™

organizes your source code, simplifies debugging, and makes documentation a snap! It lists one or more source files with informative page headings and optional line numbers, while offering invaluable features:

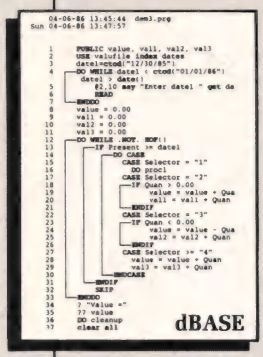


The Index (Cross-Reference list) saves you time by showing exactly where variables are used and where functions, procedures, and routines are called.





$97.00



Locations where new values may be assigned to variables are shown, making it easy to track down that mysterious value change.

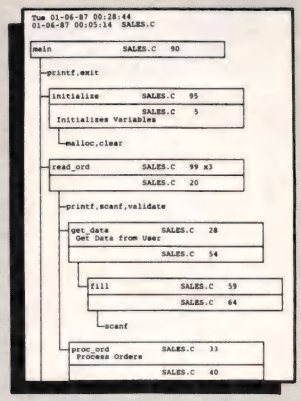
Structure Outlining solves the problem of hard-to-see nested control structures by automatically drawing lines around them.

Automatic Indentation of source code and listings reduces your editing time and ensures indentation accuracy.

Plus . . . Source Print generates a table of contents listing functions and procedures. Keywords can be printed in boldface on most printers. Multi-statement BASIC lines can be split for readability. Functions and procedures can be drawn by name from one or more source files to form a new file.

Tree Diagrammer™

shows your program's overall organization at a glance. Ordinary program listings merely display functions, procedures, and subroutines sequentially, but do not display the relationships between these routines. Our revolutionary new Tree Diagrammer automatically creates an "organization chart" of your program showing the hierarchy of calls to functions, procedures, and subroutines. Recursive calls are indicated and designated comments in the source code will appear on the chart.



Tree Diagrammer helps you organize your program more logically. And you'll be amazed at how easy it is to debug when you see how your routines interact.

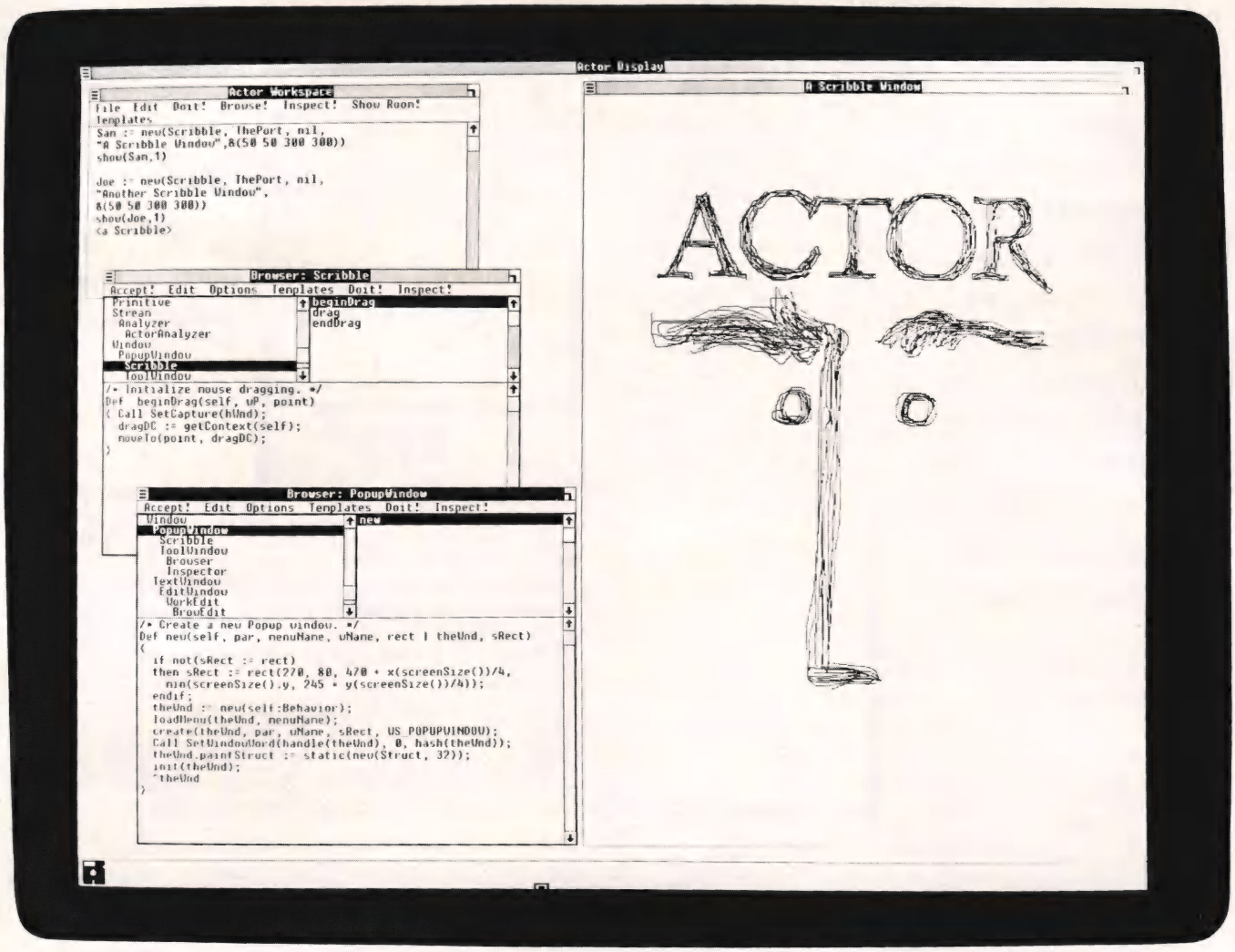
$77.00

Aldebaran Laboratories 3339 Vincent Rd. Pleasant Hill, CA 94523 415-930-8966

YES! Rush me ☐ Source Print @ $97. _____ ☐ Tree Diagrammer @ $77. _____
☐ Both $155. Ship/Handling $5. For CA add 6% tax _____ Total _____
Name _____
Company _____
Address _____
City _____ State _____ Zip _____
☐ Check enclosed ☐ VISA ☐ MasterCard ☐ American Express
Card # _____ Exp. Date _____
Signature _____ Phone # _____ 58

CIRCLE 350 ON READER SERVICE CARD

HOW TO WRITE A WINDOWS APPLICATION IN TEN MINUTES.



Actor™ is a new language that combines Microsoft® Windows with object-oriented programming. This means you can produce mouse and window applications very quickly.

For example, we created a simple "paint" program, and used it to draw the Actor logo you see on the screen. The whole program only took ten lines and ten minutes. Part of it is in the middle window on the left.

Above, you see the commands that initialized the paint window and made it appear on the screen. Below, some code that's built into Actor, specifying window behavior. Through a process known as "inheritance," it's called into play automatically.

Try programming in this new way, and you'll never go back.

Find out about Actor. Call The Whitewater Group®, (312) 491-2370.

Technology Innovation Center
906 University Place, Evanston, IL 60201

MICROSOFT® WINDOWS

CIRCLE 282 ON READER SERVICE CARD

(continued from page 48)

Implementing a Device

Once you've completed the DOS device start-up code, you need to write the functions that actually implement the device. Each device will have its own unique requirements, and the implementor must be on intimate terms with the device to be controlled. The device presented here is extremely simple as devices go—being a printer it is a character device that has output functions only. The start-up code does not change (except for the attribute word and device name, both in the device header) with the complexity of the specific device, however.

After the start-up code is set up for calling C compiled functions, it calls *driver_functions(cmd_code)*, passing the command code from the request header. *Driver_functions( )* is in drvfunc.c (Listing Three, page 72). The function *driver_functions( )* is implemented as a finite state machine that calls the device implementation functions indirectly. *Function_table* is declared in drvfunc.c as an array of pointers to functions returning void, and this array is initialized to contain pointers to those functions that this device implements. Slots for unused functions are initialized to a pointer to *bad_cmd( )*, a function that sets the request header status word to indicate an invalid command (see error.c, Listing Four, page 74). The command code is the state that selects which function to call.

For those not familiar with an array of pointers to functions, here's an explanation: Each element in the array is a pointer to, or the address of, a function that (for an 8086 small model) is an offset into the code segment. The index into the array determines which address to call, just as the index into an array of integers determines which integer to access. If you have the address of a function, you can call it indirectly as *(void)(* function_pointer)( );*, and by extending this to an array, you get *(void)(* function_pointer_array[index])( );*.

There's nothing really esoteric about it—assembly-language programmers call it a jump table. This construct produces efficient code, which is what you want in a driver.

(continued from page 51)

The alternative is a switch statement or a series of *if . . . else* pairs that compile into a lot of compare/jump instructions. Note that all the functions to which pointers are placed in the array must take the same number of parameters or chaos will result. In C, a function name (without the parentheses) evaluates to a pointer to that function.

All DOS device drivers must implement an initialization function—called only once when the driver is first loaded. Mine is called *init( )* and is found in init.c (Listing Five, page 74). The initialization function for a character device must return to DOS a long (segment:offset) pointer to the end of the driver and set the done bit in the status word. The ending address of the driver is the segment address obtained by a call to *show_cs( )* (in show_cs.asm, Listing Six, page 75) and a short pointer (offset only) to an unsigned integer called *_Uend*. *_Uend* is inserted by the Aztec linker, ln, at the end of the uninitialized data segment. Those using other compilers/linkers need to replace this by creating another module called end.c or whatever with a global unsigned called *_Uend* and make this the last module in the list when you link. My driver also displays a message on the screen, resets the parallel port, and sets the printer font to elite (12 characters/inch). For a different device, the initialization will be quite different. If this were a block device driver, in addition to all this, the initialization function would have to set the number of units and return a long pointer to an array of BPBs (BIOS parameter blocks).

All device drivers must include an initialization function.

Being a printer, the primary function this device performs is *output( )*, found in output.c (Listing Seven, page 75). Here you get from the request header the number of bytes to transfer and the segment:offset at which you find the first byte. For each character, the output function gets the character, sends it to the parallel port by calling *char_2_prn( )* (in prlport.asm, Listing Eight, page 77), checking for an error return status, and finally incrementing the transfer offset and character transferred count. If an error condition is returned by *char_2_prn( )*, the error handler function *error( )* is called; otherwise, when you have finished, you set the done bit in the request header status word. Again, this is a simple device, but you apply the same technique to develop drivers for more complex devices.

Return Info

Drivers return status information to DOS via the status word in the request header. This is represented by an unsigned integer called *status* in the request header structure *req_hdr* (see rh.h). Bit 15 is the error bit, and it is set to 1 if the driver needs to report an error condition to DOS, with the 8 least significant bits holding the error code. Code to implement an error condition is in error.c, and for my printer driver, the only possible error codes are those that are returned by the parallel port BIOS. This function is called only when errors occur; otherwise, the driver functions set bit 8, the done bit, to 1 indicating to DOS that the device has completed its task successfully.

Linking

Linking the driver is different from linking a normal C program. You must make all the logical segments fall into the same physical segment, and the device header must be at offset 0 in the file. One of the reasons why I use Aztec C is because its linker, ln, can accomplish all the preceding tasks in response to a few command-line arguments, making it a useful and powerful tool for dealing with the 8086 architecture.

To link the driver using ln, you type:

```
ln -t -b 0 -c 0 -o prndrv.com
   prndrv.o [list of files and libraries]
```

where *-t* saves the symbol table in prndev.sym, *-b 0* sets the base address to 0h (same as *ORG 0*), *-c 0* makes the code segment start at offset 0h in the physical segment, and *-o prndrv.com* means that the output file is prndrv.com (specifying an extension of .COM causes all logical segments to be in one physical segment).

Again, users of other compilers/linkers will have to adapt this to their systems. Hint: Try using a *GROUP* statement in prndrv.asm to cause the code and data segments to be joined into one physical segment, and remember to take the offset into the group instead of into the segment when using the offset operator.

Known Shortcomings

All variables declared outside a function must be initialized to some value—they cannot be assumed to be initialized to 0, and if they are not initialized, the scheme gets buggy. Drivers written in C tend to be large—probably all those references to *bp*.

Availability

All the source code for articles in this issue is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal*, 501 Galveston Dr., Redwood City, CA 94063, or call (415) 366-3600, ext. 216. Please specify the issue number and format (MS-DOS, Macintosh, Kaypro).

DDJ

(Listings begin on page 68.)

Vote for your favorite feature/article.
Circle Reader Service **No. 4.**

Listing One *(Text begins on page 28.)*

```
/*
** Copyright (c) 1987,  Tom Steppe.  All rights reserved.
**
** This module compares two arrays of lines (representing
** files) and reports the sequences of consecutive matching
** lines between them using the "recursive longest matching
** sequence" algorithm.  This is useful for implementing a
** file comparison utility.
**
** Compiler:  Microsoft (R) C Compiler  Version 4.00
*/

#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <malloc.h>

/* Boolean type and values. */
typedef int     BOOLEAN;
#define TRUE    1
#define FALSE   0

/* Minimum macro. */
#define min(x, y)    (((x) <= (y)) ? (x) : (y))

/* Value to indicate identical strings with strcmp. */
#define ALIKE   0

/* Result of hashing function for a line of text. */
typedef unsigned int   HASH;

/* Mask for number of bits in hash code.  (12 bits). */
#define MASK    (unsigned int) 0x0FFF

/* Number of possible hash codes. */
#define HASHSIZ    (MASK + 1)

/* Information about an entry in a hash table. */
typedef struct tblentry
{
    int   frst;   /* First line # with this hash code. */
    int   last;   /* Last line # with this hash code. */
} TBLENTRY;

/* Information about a line of text. */
typedef struct lineinf
{
    HASH   hash;    /* Hash code value. */
    int    nxtln;   /* Next line with same hash (or 0). */
} LINEINF;

/* Information about a file. */
typedef struct fileinf
{
    char       **txt;      /* Array of lines of text. */
    LINEINF    *line;      /* Array of line info structs. */
    TBLENTRY   *hashtbl;   /* Hash table. */
} FILEINF;

/* Function declarations. */
BOOLEAN filcmp      (char **, int, char **, int, int);
BOOLEAN get_inf     (char **, int, FILEINF *);
HASH    calc_hash   (char *);
void    fnd_seq     (FILEINF *, int, int,
                          FILEINF *, int, int, int);
BOOLEAN chk_hashes  (LINEINF *, LINEINF *, int);
int     cnt_matches (char **, char **, int);
void    rpt_seq     (int, int, int);

/*********************************************************
** compare compares two arrays of lines and reports the
** sequences of consecutive matching lines.  The zeroth
```

```
** element of each array is unused so that the index into
** the array is identical to the associated line number.
**
** RETURNS:  TRUE if comparison succeeded.
**           FALSE if not enough memory.
*************************************************************/

BOOLEAN compare (a1, n1, a2, n2, lngval)

char    **a1;      /* (I) Array of lines of text in #1. */
int     n1;        /* (I) Number of lines in a1.
                          (Does not count 0th element.) */
char    **a2;      /* (I) Array of lines of text in #2. */
int     n2;        /* (I) Number of lines in a2.
                          (Does not count 0th element.) */
int     lngval;    /* (I) "Long enough" value. */
{
    FILEINF   f1;     /* File information for #1. */
    FILEINF   f2;     /* File information for #2. */
    BOOLEAN   rtn;    /* Return value. */

    /* Gather information for each file, then compare. */
    if (rtn =
        (get_inf (a1, n1, &f1) && get_inf (a2, n2, &f2)))
    {
        fnd_seq (&f1, 1, n1, &f2, 1, n2, lngval);
    }

    return (rtn);
}

/*************************************************************
** get_inf calculates hash codes and builds a hash table.
**
** RETURNS:  TRUE if get_inf succeeded.
**           FALSE if not enough memory.
*************************************************************/

static BOOLEAN get_inf (a, n, f)

char      **a;    /* (I) Array of lines of text. */
int       n;      /* (I) Number of lines in a. */
FILEINF   *f;     /* (O) File information. */
{
    unsigned int   size;     /* Size of hash table. */
    register int   i;        /* Counter. */
    TBLENTRY       *entry;   /* Entry in hash table. */

    /* Assign the array of text. */
    f->txt = a;

    /* Allocate and initialize a hash table. */
    size = HASHSIZ * sizeof (TBLENTRY);
    if (f->hashtbl = (TBLENTRY *) malloc (size))
    {
        memset ((char *) f->hashtbl, '\0', size);
    }
    else
    {
        return (FALSE);
    }

    /* If there are any lines: */
    if (n > 0)
    {
        /* Allocate an array of line structures. */
        if (f->line = (LINEINF *)
                malloc ((n + 1) * sizeof (LINEINF *)))
        {
            /* Loop through the lines. */
            for (i = 1; i <= n; i++)
            {
```

(continued on next page)

Listing One *(Listing continued, text begins on page 28.)*

```
                /* Calculate the hash code value. */
                f->line[i].hash = calc_hash (f->txt[i]);

                /* Locate the entry in the hash table. */
                entry = f->hashtbl + f->line[i].hash;

                /* Update the linked list of lines with */
                /* the same hash code.                  */
                f->line[entry->last].nxtln = i;
                f->line[i].nxtln = 0;

                /* Update the first and last line */
                /* information in the hash table. */
                if (entry->frst == 0)
                {
                    entry->frst = i;
                }
                entry->last = i;
            }
        }
        else
        {
            return (FALSE);
        }
    }
    else
    {
       f->line = NULL;
    }

    return (TRUE);
}

/*************************************************************
** calc_hash calculates a hash code for a line of text.
**
** RETURNS:  a hash code value.
*************************************************************/

static HASH calc_hash (buf)

char   *buf;   /* (I) Line of text. */
{
    register unsigned int   chksum;   /* Checksum. */
    char                    *s;       /* Pointer. */
    HASH                    hash;     /* Hash code value. */

    /* Build up a checksum of the characters in the text. */
    for (chksum = 0, s = buf; *s; chksum ^= *s++)
    {
        ;
    }

    /* Combine the 7-bit checksum and as much of the */
    /* length as is possible.                        */
    hash = ((chksum & 0x7F) | ((s - buf) << 7)) & MASK;

    return (hash);
}

/*************************************************************
** Given starting and ending line numbers, fnd_seq finds a
** "good sequence" of lines within those ranges.  fnd_seq
** then recursively finds "good sequences" in the sections
** of lines above the "good sequence" and below it.
*************************************************************/

static void fnd_seq (f1, beg1, end1, f2, beg2, end2, lngval)

FILEINF   *f1;      /* (I) File information for #1. */
int       beg1;     /* (I) First line # to compare in #1. */
int       end1;     /* (I) Last line # to compare in #1. */
```

```
FILEINF   *f2;      /* (I) File information for #2. */
int       beg2;     /* (I) First line # to compare in #2. */
int       end2;     /* (I) Last line # to compare in #2. */
int       lngval;   /* (I) "Long enough" value. */
{
    LINEINF      *line1;   /* Line information ptr in #1. */
    LINEINF      *line2;   /* Line information ptr in #2. */
    register int limit;    /* Looping limit. */
    int          ln1;      /* Line number in #1. */
    int          ln2;      /* Line number in #2. */
    register int ln;       /* Working line number. */
    BOOLEAN      go;       /* Continue to loop? */
    int          most;     /* Longest possible seq. */
    int          most1;    /* Longest possible due to #1. */
    int          most2;    /* Longest possible due to #2. */
    int          cnt;      /* Length of longest seq. */
    int          oldcnt;   /* Length of prev longest seq. */
    int          n;        /* Length of cur longest seq. */
    int          m1;       /* Line of longest seq. in #1. */
    int          m2;       /* Line of longest seq. in #2. */

    /* Initialize. */
    go    = TRUE;
    line1 = f1->line;
    line2 = f2->line;

    /* Initialize longest sequence information. */
    cnt    = 0;          /* Length of longest seq. */
    m1     = beg1 - 1;   /* Line # of longest seq. in #1. */
    m2     = beg2 - 1;   /* Line # of longest seq. in #2. */
    oldcnt = 0;          /* Length of prev longest seq. */

    /* Calculate maximum possible number of consecutive */
    /* lines that can match (based on line # ranges).   */
    most1 = end1 - beg1 + 1;
    most2 = end2 - beg2 + 1;

    /* Scan lines looking for a "good sequence".
    ** Compare lines in the following order of line numbers:
    **
    **                  (1, 1)
    **          (1, 2), (2, 1), (2, 2)
    ** (1, 3), (2, 3), (3, 1), (3, 2), (3, 3)
    ** etc.
    */
    for (ln1 = beg1, ln2 = beg2; TRUE; ln1++, ln2++)
    {
        if (ln2 <= end2 - cnt)
        /* There are enough lines left in #2 such that it */
        /* is possible to find a longer sequence.         */
        {
            /* Determine the limit in #1 that both       */
            /* enforces the order scheme and still makes */
            /* it possible to find a longer sequence.    */
            limit = min (ln1 - 1, end1 - cnt);

            /* Calculate first potential match in #1. */
            for (ln = f1->hashtbl[line2[ln2].hash].frst;
                    ln && ln < beg1; ln = line1[ln].nxtln)
            {
               ;
            }

            /* Loop through the lines in #1. */
            for (; ln && ln <= limit; ln = line1[ln].nxtln)
            {
                if (line1[ln].hash == line2[ln2].hash &&
                        line1[ln + cnt].hash ==
                            line2[ln2 + cnt].hash &&
                        !(ln - m1 == ln2 - m2 &&
                        ln < m1 + cnt && m1 != beg1 - 1))
                /* A candidate for a longer sequence has */
```

(continued on next page)

Listing One *(Listing continued, text begins on page 28.)*

```
            /* been located.  The current lines      */
            /* match, the current lines + cnt match, */
            /* and this sequence is not a subset of  */
            /* the longest sequence so far.          */
            {
                /* Calculate most possible matches. */
                most = min (end1 - ln + 1, most2);

                /* First compare hash codes.  If the  */
                /* number of matches exceeds the      */
                /* longest sequence so far, then      */
                /* compare the actual text.           */
                if (chk_hashes (line1 + ln,
                          line2 + ln2, cnt) &&
                        (n = cnt_matches (f1->txt + ln,
                            f2->txt + ln2, most)) > cnt)
                /* This is the longest seq. so far. */
                {
                    /* Update longest sequence info. */
                    oldcnt = cnt;
                    cnt    = n;
                    m1     = ln;
                    m2     = ln2;

                    /* If it's long enough, end the */
                    /* search.                      */
                    if (cnt >= lngval)
                    {
                        break;
                    }

                    /* Update limit, using new count. */
                    limit = min (ln1 - 1, end1 - cnt);
                }
            }
        }

        /* If it's long enough, end the search. */
        if (cnt >= lngval)
        {
            break;
        }
        most2--;
    }
    else
    {
        go = FALSE;   /* This file is exhausted. */
    }

    /* Repeat the process for the other file. */
    if (ln1 <= end1 - cnt)
    {
        limit = min (ln2, end2 - cnt);

        for (ln = f2->hashtbl[line1[ln1].hash].frst;
                ln && ln < beg2; ln = line2[ln].nxtln)
        {
           ;
        }

        for (; ln && ln <= limit; ln = line2[ln].nxtln)
        {
            if (line1[ln1].hash == line2[ln].hash &&
                    line1[ln1 + cnt].hash ==
                        line2[ln+ cnt].hash &&
                    !(ln1 - m1 == ln - m2 &&
                    ln1 < m1 + cnt && m2 != beg2 - 1))
            {
                most = min (end2 - ln + 1, most1);

                if (chk_hashes (line1 + ln1,
                            line2 + ln, cnt) &&
```

```
                        (n = cnt_matches (f1->txt + ln1,
                            f2->txt + ln, most)) > cnt)
                    {
                        oldcnt = cnt;
                        cnt    = n;
                        m1     = ln1;
                        m2     = ln;

                        if (cnt >= lngval)
                        {
                            break;
                        }

                        limit = min (ln2, end2 - cnt);
                    }
                }
            }

            if (cnt >= lngval)
            {
                break;
            }
            most1--;
        }
        else if (!go)
        {
            break;   /* This file is exhausted, also. */
        }
    }

    /* If the longest sequence is shorter than the "long */
    /* enough" value, the "long enough" value can be     */
    /* adjusted for the rest of the comparison process.  */
    if (cnt < lngval)
    {
        lngval = cnt;
    }

    if (cnt >= 1)
    /* Longest sequence exceeds minimum necessary size. */
    {
        if (m1 != beg1 && m2 != beg2 && oldcnt > 0)
        /* There is still something worth comparing */
        /* previous to the sequence.                */
        {
            /* Use knowledge of the previous longest seq. */
            fnd_seq (f1, beg1, m1 - 1,
                    f2, beg2, m2 - 1, oldcnt);
        }

        /* Report the sequence. */
        rpt_seq (m1, m2, cnt);

        if (m1 + cnt - 1 != end1 && m2 + cnt - 1 != end2)
        /* There is still something worth comparing */
        /* subsequent to the sequence.              */
        {
            fnd_seq (f1, m1 + cnt, end1,
                    f2, m2 + cnt, end2, lngval);
        }
    }
}

/*********************************************************
** chk_hashes determines whether this sequence of matching
** hash codes is longer than cnt.  It knows that the first
** pair of hash codes is guaranteed to match.
**
** RETURNS:  TRUE if this sequence is longer than cnt.
**           FALSE if this sequence is not longer than cnt.
*********************************************************/
```

(continued on next page)

Listing One *(Listing continued, text begins on page 28.)*

```
static BOOLEAN chk_hashes (line1, line2, cnt)

LINEINF         *line1;   /* (I) Line information for #1. */
LINEINF         *line2;   /* (I) Line information for #2. */
register int    cnt;      /* (I) Count to try to exceed. */
{
    register int   n;   /* Count of consecutive matches. */

    for (n = 1; n <= cnt &&
        ((++line1)->hash == (++line2)->hash); n++)
    {
        ;
    }

    return (n > cnt);
}

/*********************************************************
** cnt_matches counts the number of consecutive matching
** lines of text.
**
** RETURNS:  number of consecutive matching lines.
*********************************************************/

static int cnt_matches (s1, s2, most)

char          **s1;   /* (I) Starting line in file #1. */
char          **s2;   /* (I) Starting line in file #2. */
register int most;   /* (I) Most matching lines possible. */
{
    register int   n;   /* Count of consecutive matches. */

    /* Count the consecutive matches. */
    for (n = 0; n < most && strcmp (*s1++, *s2++) == ALIKE;
            n++)
    {
        ;
    }

    return (n);
}

/*********************************************************
** rpt_seq reports a matching sequence of lines.
*********************************************************/

static void rpt_seq (m1, m2, cnt)

int   m1;    /* (I) Location of matching sequence in #1. */
int   m2;    /* (I) Location of matching sequence in #2. */
int   cnt;   /* (I) Number of lines in matching sequence. */
{
    fprintf (stdout,
        "Matched %5d lines:  (%5d - %5d) and (%5d - %5d)\n",
         cnt, m1, m1 + cnt - 1, m2, m2 + cnt - 1);
}
```

End Listing

XOR CHAIN

Listing One *(Text begins on page 36.)*

```
/* Binary tree delete procedure
 *
 * Input parameter "s" points to a scan for an XOR
 *   chained binary tree.
 * The procedure deletes s->child from the tree,
 *   and returns with s set by GoParent(s)
 */

void Delete(s)
struct Scan *s;
{
    struct Scan temp, *t;
    struct Item *i, *j, *k;

    i = s->child;   /*i is the item to be deleted*/
    GoParent(s);

    if (i->llink == i->rlink)          /* Case 1 */
    {
        /*
         * adjust the pointers for s->child,
         * i's parent
         */
        if (i->key < s->child->key)
            s->child->llink = s->parent;
        else
            s->child->rlink = s->parent;
    }

    else if (i->llink == s->child)     /* Case 2 */
    {
        /*
         * adjust the pointers for s->child,
         * i's parent
         */

        if (i->key < s->child->key)
            s->child->llink = i->rlink  ^
                              s->parent ^ s->child;
        else
            s->child->rlink = i->rlink  ^
                              s->parent ^ s->child;

        /*
         * adjust the pointers for i's child
         */

        j = i->rlink ^ s->child;
        j->rlink ^= i ^ s->child;
        j->llink ^= i ^ s->child;
    }

    else if (i->rlink == s->child)     /* Case 3 */
    {
        /*
         * adjust the pointers for s->child,
         * i's parent
         */
        if( i->key < s->child->key )
            s->child->llink = i->llink  ^
                              s->parent ^
                              s->child;
        else
            s->child->rlink = i->llink  ^
                              s->parent ^
                              s->child;
        /*
         * adjust the pointers for i's child
         */

        j = i->llink ^ s->child;
        j->rlink ^= i ^ s->child;
```

(continued on page 65)

Create 68K Embedded Systems On ATs

Oregon Software Brings VAX and VME Pascal-2® Cross-Development Tools To MS-DOS Workstations

Get All Your Tools In One Package

The Pascal-2 Cross-Development System gives you price/performance value that beats high-priced workstations. Compare these features and call for further information.

Compiler

- 68000, 68020, and 68881 instruction sets
- ROMable code
- reentrant code
- separate program sections for code and data
- easy access to hardware through fixed memory locations (I/O space)

Assembler-Linker

- VERSAdos compatible output
- easily linked, relocatable S-record format

Concurrent Program Support

- process synchronization primitives for creating a multiprocessing real-time kernel
- interactive post-mortem analyzer
- software-selectable interrupt levels

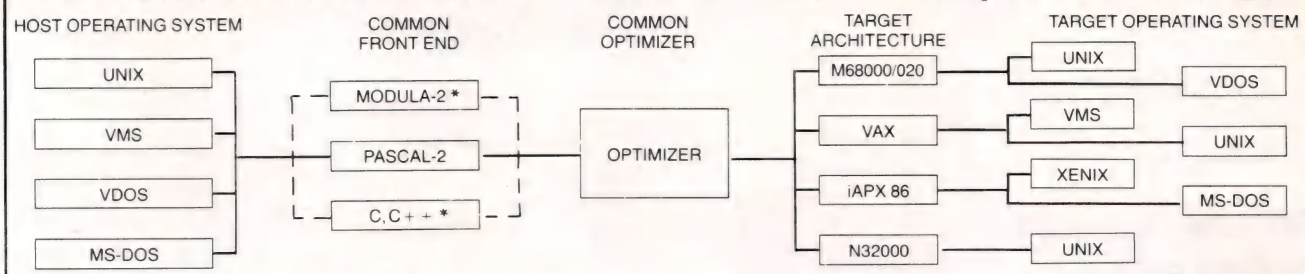
Stand-Alone Library

- source supplied: users may adapt code as needed
- user programs that run without an operating system or other real-time kernel
- compact code for extremely small ROM programs or limited memory devices
- ability to write device drivers in Pascal or assembly code
- user-customized system services or control programs
- easily-gathered performance analysis statistics

Source Management

- a version-control system to track and identify each phase of development
- optional branching to multiple versions from a single source
- automated rebuilding of programs from component modules (a Unix-style Make function)
- a restricted-access system for coordinating revisions

Ten Years of Sustained and Focused Growth in Compiler Technology



*Release planned in 1988.

OREGON SOFTWARE

800-367-2202

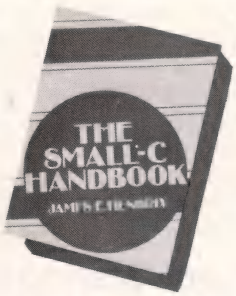
6915 SW Macadam Avenue, Portland, Oregon 97219 U.S.A. 503-245-2202 FAX 503-245-8449 TWX 910-464-4779

DISTRIBUTORS: ■ **DENMARK** erik mainz a/s, Theklavej 46, DK-2400 Copenhagen NV, Tel: 1-34-7788 ■ **ENGLAND** Real Time Products, 1 Paul Street, London EC2A 4JJ, Tel: 1-588-0667; Grey Matter, Microcomputer Software and Consultancy, 4, Prigg Meadow, Ashburton, Devon TQ13 7DF, Tel: 364-53499 ■ **FRANCE** Focal Informatique, 5 Place de la Gare, Le Rhodanien niveau 7, 69003 Lyon Part-Dieu, Tel: 72-33-02-02; SCT Electronique, Za Route du Bua, Batiment B-Entree 2, Cidex 434 Verrieres Le Buisson, Tel: 1-6011 1950 ■ **THE NETHERLANDS** Computing & Systems Consultants BV, Stationsplein 47, 5611 BC Eindhoven, 040434957 ■ **SWEDEN** IND AB, Box 2066, S-175 02 Jaerfalla, Tel: 75852025 ■ **SWITZERLAND** Fabrimex AG Kirchenweg 5, 8032 Zurich, Tel: 01-2512929; Muhlethaler AG Computer Systems Kirchstrasse 2, CH-4536 Attiswil, Tel: 65772911 ■ **WEST GERMANY** AC Copy Kurbrunnenstrasse 30, Aachen D-5100, Tel: 241-505477.

CIRCLE 357 ON READER SERVICE CARD

DR. DOBB'S C TOOLBOX

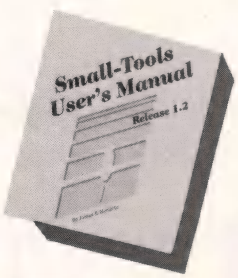
Small-C Handbook & Small-C Compiler



This compiler and handbook provide everything you need for learning how compilers are constructed, and for learning C at its most fundamental level. You'll find a discussion of assembly language concepts and program translation tools, and of how to generate a new version of the compiler. Full source code is included. Please specify MS/PC-DOS or CP/M: Kaypro, Osborne, Apple, Zenith Z-100 DS/DD, 8" SS/SD.

CP/M Compiler & Handbook	Item #006B	$37.90
MS/PC-DOS Compiler & Handbook	Item #006C	$42.90

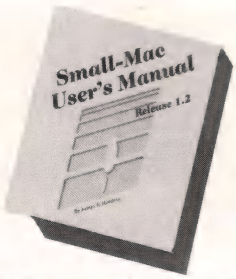
Small-Tools: Programs for Text Processing



This package of Small C programs performs specific, modular operations on text files. It is supplied as source code with full documentation. With the Small-C Compiler, you can select and adapt these tools to meet your needs. Please specify MS/PC-DOS or CP/M: Kaypro, Osborne, Apple, Zenith Z-100 DS/DD, 8" SS/SD.

Small-Tools	Item #010A	$29.95

Small-Mac: An Assembler for Small-C



This package includes: a simplified macro facility, C language expression operators, object file visibility, descriptive error message and an externally defined instruction table. Source code and documentation is included. CP/M only. Please specify: Kaypro, Osborne, Apple, Zenith Z-100 DS/DD, 8" SS/SD.

Small-Mac	Item #012A	$29.95

Special Packages

CP/M C Package
Save Over $27!

Receive: Dr. Dobb's Toobook of C, The Small-C Handbook and Small-C Compiler, Small-Mac Assembler, and Small-Tools Text Processing Programs. Only $99.95!

CP/M Package	Item #005A	$99.95

MS/PC-DOS C Package
Save $22

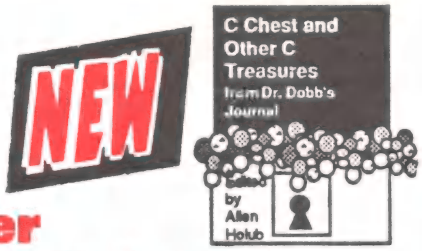
Receive: Dr. Dobb's Toolbook of C, The Small-C Handbook and MS/PC-DOS Addendum, Small-C Compiler, Small-Tools Texts Processing Programs and Small Windows. Only $109.95!

Specify Microsoft C Version 4.0, Small C or Lattice C compiler.

MS/PC-DOS Package	Item #005W	$109.95

C Disk Formats
Please indicate MS/PC-DOS or CP/M. For CP/M specify: Apple, Osborne, Kaypro, Zenith Z-100 DS/DD, 8" SS/SD.

For Small-Windows, specify Small-C, Microsoft C Version 4.0 or Lattice C compiler.



C Chest and Other C Treasures from Dr. Dobb's Journal

This comprehensive anthology contains the popular "C Chest" columns from *Dr. Dobb's*, along with the lively philosophical and practical discussions they inspired. You'll also find other information-packed articles by C experts.

Topics covered include: pipes, wild-card expansion, and quoted arguments; sorting routines; command-line processing; queues and bit maps; utilities such as *ls*, *make*, and *more*; expression parsing; hypenation; IBM cursor control and an Fget that edits; redirection; accessing IBM video display memory; trees; an AVL tree database package; directory traversal; sets; shrinking. EXE file images; hashing, expressions, and Roman numerals; and statistical applications of digital low-pass filters.

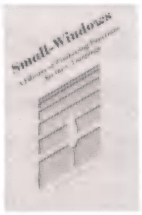
Other C treasures include: a variable metric minimizer; Christensen protocols; Fgrep; a peephole optimizer; curve fitting with cubic splines; and the CompuServe B protocol.

All subroutines and programs are written in C and are available on disk with full source code. MS-DOS format.

Book & MS-DOS Disk	Item #49-6	$39.95
Book only	Item #40-2	$24.95

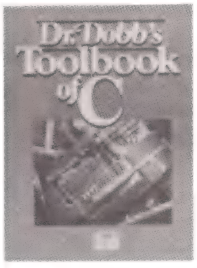
NOW FOR MICROSOFT C, SMALL C, AND LATTICE C COMPILERS

Small-Windows: A Library of Windowing Functions for the C Language



Small-Windows is a complete windowing library for C. The package contains: 18 video functions written in assembly language; 7 menu functions that support both static and pop-up menus; and 41 window functions, including functions to clean, frame, move, hide, show, scroll, push, and pop windows. A file directory facility illustrates the use of the window menu functions and provides file selection, renaming, and deletion capability. Two test programs are also included. For PC/MS-DOS systems, Microsoft C Version 4.0, Small-C, and Lattice C compilers. Documentation and full source code is included.

Small-Windows	Item #35-6	$29.95



Dr. Dobb's Toolbook of C

This authoritative reference contains over 700 pages, including the best C article from Dr. Dobb's Journal along with new material. You'll find hundreds of pages of valuable C source code, including a complete compiler, an assembler, and text processing programs.

Toolbook of C	Item #005	$29.95

ORDER FORM

TO ORDER:
Return this order form with your payment to:
M&T Books
501 Galveston Dr.
Redwood City, CA
94063
Or, call
TOLL-FREE
800-533-4372
Mon-Fri 8AM-5PM
In CA call
800-356-2002

Name _____

Address _____

City _____ State _____ Zip _____

Item # Description Price

_____

_____

_____

_____

Subtotal _____

CA residents add sales tax ___ % _____

Add $2.25 per item for shipping _____

TOTAL _____

For Disk Orders, please indicate format. Refer to product description for standard format availability.

___ MS-DOS CP/M ___ Kaypro ___ 8" SS/SD
___ Osborne ___ Apple ___ Zenith Z-100 DS/DD

For *Small-Windows*, indicate:
___ Microsoft version 4.0 compiler
___ Small-C compiler
___ Lattice C compiler

___ Check enclosed. Make payable to M&T Publishing.

Charge my ___ VISA ___ M/C ___ Amer. Exp.
Card No. _____ Exp. _____
Signature _____

CODE: 3131A

MS-DOS WITH UP-STEP

Taming MS-DOS

by Thom Hogan

Taming MS-DOS takes you beyond the basics, picking up where your DOS manual leaves off. You'll learn how to create a memory-resident clock, how to rename subdirectories and change file attributes, how to create AUTOEXEC.BAT files, and how to customize CONFIG.SYS and use ANSI.SYS to change the appearance of DOS. You'll find extensive batch file coverage with example routines that use redirection operators, filters and pipes, and ready-to-use assembly language programs that enhance DOS. Full source code is included.

Taming MS-DOS Item #24-0 $19.95
Taming MS-DOS with disk Item #59-3 $34.95

On Command: Writing a Unix-Like Shell for MS-DOS

by Allen Holub

This book and ready-to-use program demonstrate how to write a Unix-like shell for MS-DOS, with techniques applicable to most other programming environments as well. The book and disk include a detailed description and working version of the shell, complete C source code, a thorough discussion of low-level DOS interfacing and significant examples of C programming at the system level. Supported features: read, aliases, history and C-Shell-based shell scripts. The Unix-like control flow includes: if/then/else; while; foreach; switch/case; break; continue. For IBM PC and direct compatible's. All source code included on disk.

On Command Item #29-1 $39.95

/Util

When used with the **shell,** this collection of utility programs and subroutines provide you with a fully functional subset of the Unix environment. Utilities include: cat; cp; date; du; echo; grep; ls; mkdir; mv; p; pause; printevn; rm; rmdir; sub; and chmod. Complete source code and manual included.

/Util Item #12-7 $29.95

Program Interfacing to MS-DOS

by William Wong

Originally featured in *Micro/Systems Journal,* **Program Interfacing to MS-DOS** provides ten concise articles that will orient any experienced programmer to the MS-DOS environment. All source code discussed is also contained on disk.

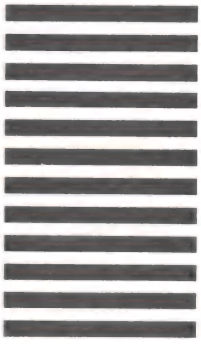
Topics include: program construction, character base input and output functions, and file access. You'll also find a discussion of CP/M style vs. Unix-style DOS file access, sample program files, and a detailed description of how to build device drivers. A device driver for a memory disk and a character device driver are provided on disk with full source code.

Interfacing to MS-DOS Item #34-8 $29.95

NR is a text formatter that is written in C and is compatible with the Unix NROFF. It includes complete implementation of the -ms macro package, and an in-depth description of how -ms works. **NR** does hyphenation and simple proportional spacing, and supports automatic table of contents generation and indexing, automatic footnotes and endnotes, italics, boldface, overstricking, understricking, and left and right margin adjustment. Also: extensive macro and string capability, number registers in various formats, diversions and diversion traps, input and output line traps. Full source code included. For PC compatibles.

NR Item #33-X $29.95

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES



BUSINESS REPLY MAIL
FIRST CLASS PERMIT 871 REDWOOD CITY, CA

POSTAGE WILL BE PAID BY ADDRESSEE

M&T Books
501 Galveston Dr.
Redwood City, CA 94063

Listing One *(Listing continued, text begins on page 36.)*

```
    j->llink ^= i ^ s->child;
}

else                                    /* Case 4 */
{
    j = i->llink ^ s->child;

    if (j->rlink == i)                  /* Case 4a */
    {
        /*
         * adjust the pointers for s->child,
         * i's parent
         */

        if (i->key < s->child->key)
            s->child->llink = j ^ s->parent;
        else
            s->child->rlink = j ^ s->parent;

        /*
         * adjust the pointers for i's children
         */

        j->llink ^= i ^ s->child;
        j->rlink  = i->rlink;
        k = i->rlink ^ s->child;
        k->llink ^= i ^ j;
        k->rlink ^= i ^ j;
    }

    else                                /* Case 4b */
    {
```

(continued on next page)

XOR CHAIN

Listing One

(Listing continued, text begins on page 36.)

```
/*
 * locate the replacement item
 */

t = &temp;
Associate(t,s->root);
t->parent = s->child;
t->child  = i;
GoLeft(t);

while( t->child->rlink != t->parent )
    GoRight(t);

/*
 * adjust the pointers to free t->child
 */

t->parent->rlink ^= t->child->llink ^
                    t->parent     ^
                    t->child;

if (t->child->llink != t->parent)
{
    k = t->child->llink ^ t->parent;
    k->llink ^= t->parent ^ t->child;
    k->rlink ^= t->parent ^ t->child;
}

/*
 * adjust the pointers for s->child,
 * i's parent
 */

if (i->key < s->child->key)
    s->child->llink = t->child ^ s->parent;
else
    s->child->rlink = t->child ^ s->parent;

t->child->llink = i->llink;
t->child->rlink = i->rlink;

            /*
             * adjust the pointers for i's children
             */

            j->llink ^= i ^ t->child;
            j->rlink ^= i ^ t->child;
            k = i->rlink ^ s->child;
            k->llink ^= i ^ j;
            k->rlink ^= i ^ j;
        }
    }
    DropItem(i);
}
```

End Listing

Turbo Tech Report Speaks Your Language.

Turbo Pascal Articles and Reviews


TURBO TECH REPORT

Turbo Watch

Turbo Tech Report will make a point to address the entire Turbo Pascal programming community, by discussing the issue of porting code across the three Turbo Pascal environments.

IN THIS ISSUE:

A disk filled with Turbo Pascal code!

News and commentary

The newsletter/disk publication for Turbo Pascal® users

Are you looking for powerful utilities written in Turbo Pascal that you can use to develop software or incorporate into your programs? Are you interested in improving and expanding your Turbo Pascal programming skills?

Then you deserve a subscription to *Turbo Tech Report*, the bimonthly newsletter/disk publication from the publishers of *Dr. Dobb's Journal* and *Micro/Systems Journal*. Each issue delivers more than 250K of Turbo Pascal source code programs on disk, and 24+ pages of articles, Turbo Pascal software and book reviews, and analysis and commentary. It's the only publication delivering such focused technical articles with code on disk. Each valuable issue contains:

- ***Articles*** on topics like speedy 3D graphics, mathematical expression parsers, creating global gotos, memory resident and AI applications and more—all written by Turbo experts.
- ***Reviews*** of the latest Turbo Pascal software programs from companies like Borland International, Blaise Computing, Media Cybernetics, Nostradamus, and more!
- ***News and commentary*** detailing the latest products and developments in the Turbo Pascal programming community.
- ***A disk filled with Turbo Pascal code!*** You'll get the Turbo Pascal utilities and routines discussed in the newsletter's articles, as well as applications developed by Turbo users from around the world. You'll receive programs that make labels, generate menus, provide faster screen access, transfer files, and more!

If you're an expert Turbo Pascal programmer or a novice interested in expanding your Turbo skills, you need a publication that speaks your language: *Turbo Tech Report*. Subscribe today at the special price of just $99—that's 33% off the regular price of $150. To order by credit card, call toll-free 1-800-533-4372. Or mail the coupon with your payment to *Turbo Tech Report*, 501 Galveston Drive, Redwood City, CA 94063.

—**Yes!** I want a one-year subscription to *Turbo Tech Report* (6 issues with 6 disks) for $99.

Format: ☐ PC/MS-DOS ☐ Macintosh

CP/M: ☐ Kaypro ☐ Osborne ☐ Apple

PAYMENT MUST ACCOMPANY ALL ORDERS

☐ Check/money order enclosed.

☐ Charge my: ___ VISA ___ M/C ___ AmExp.

Card # _____ Exp. _____

Signature _____

Name _____

Address _____

City _____ State _____ Zip _____

Orders outside U.S.: add $30.

CIRCLE 119 ON READER SERVICE CARD

Listing One *(Text begins on page 44.)*

```
;  prndrv.asm
;        printer driver startup code
;        ms/pc-dos 2.x, 3.x installable device driver
;copyright (c) Andy Klein 1987


PRIVATE_STACK_SIZ  equ  256   ;private stack size, probably much larger
                              ;  than necessary


IS_CHAR_DEV  equ 32768        ;bit to set for this driver


codeseg    segment para public 'CODE'
codeseg    ends

dataseg    segment para public 'DATA'    ; define first
dataseg    ends


    assume    cs:codeseg, ds:dataseg, es:dataseg, ss:dataseg


codeseg    segment para public 'CODE'

    org    0        ;drivers are always org'ed at 0
                    ;  with the device header first

public prn_driver, dev_strategy_, dev_interrupt_


prn_driver    proc    far  ;drivers invoked as far calls by DOS

        ;device header starts here

next_dev   db 4 dup(255)        ;no next driver in this file, need a
                                ;(long)-1 which is 4 bytes of 255
attribute  dw    IS_CHAR_DEV
strategy   dw    dev_strategy_  ;device strategy entry point
interrupt  dw    dev_interrupt_ ;device interrupt entry point
dev_name   db    'PRN     '

        ;end of the device header


    ;code segment variables, these will be addressable before we setup
    ; our data and stack segment
req_hdr_seg  dw    (0)   ; pointer to request header, segment part
req_hdr_off  dw    (0)   ; pointer to request header, offset part
caller_ss    dw    (0)   ; caller's ss
caller_sp    dw    (0)   ; caller's sp


        ;strategy - this strategy function saves a long pointer to a
        ;  request header in code segment variables req_hdr_seg and
        ;  req_hdr_off.
        ;  The address of the request header is passed in es:bx

dev_strategy_:

    mov word ptr cs:req_hdr_off,bx     ;save request header pointer
    mov word ptr cs:req_hdr_seg,es     ;  in code segment variables
    ret                                ; ...and back to dos


public $cswt, $begin    ;keep Aztec linker happy, all Aztec C compiled
                        ; functions have a reference to these 2 functions
                        ; which normally drag in the startup code

    $cswt proc near
    $cswt endp

    $begin proc near
    $begin endp
```

```
    extrn    driver_functions_:near

    ;interrupt - not a true interrupt handler (it ends with a ret as
                                                          opposed
    ; to an iret), this function recieves no parameters.  Instead it
    ; uses the information stored in the request header we recieved
                                                            and
    ; saved a pointer to in the strategy function to determine what
                                                          to do.

dev_interrupt_:
            ;STEP 0
            ; Preserve machine state
    cli             ; no interrupts when dealing with machine state
    push ds         ; save machine state on caller's stack
    push es
    push ax
    push bx
    push cx
    push dx
    push si
    push di
    push bp
    pushf
    mov cs:caller_ss,ss     ; save caller stack frame (ss and sp)
    mov cs:caller_sp,sp
    sti                     ;interrupts OK

            ;Now the real work starts ...

            ;STEP 1
            ; Get the driver's segment into the ax register
            ; This begins our task of establishing addressability

    mov ax,cs

            ;STEP 2
            ;  Copy the request header stored at req_hdr_seg:req_hdr
                                                          _off
            ;  into our private and C addressable request header

    mov si,cs:req_hdr_off       ;load up the source string segment:
                                                          offset
    mov bx,cs:req_hdr_seg       ; into ds:si registers
    mov ds,bx
    mov es,ax                   ;load up the destination string
                                                    segment:offset
    mov di,offset cs:driver_rh_ ; into es:di registers
    cld                         ;set direction flag to increment
                                                        si and di
    xor cx,cx                   ;clear out cx
    mov cl,[si]                 ;first byte of request header is its
                                                          length
    rep movsb                   ; ... and copy it ...


            ;STEP 3
            ;  This step establishes addressability of data segment
            ;  and sets up driver's stack frame.
            ;  It also sets register bp equal to sp, a state required
            ;  for calling the first C function
            ;       Remember that register ax contains the driver
                                                          segment
            ;       We will set ds, es, ss all equal to cs ... 8080
                                                          model??
    mov bx,offset c_stack_top    ;this will go into sp
    cli             ;no interrupts while we mess with these registers
    mov ds,ax
    mov es,ax
    mov ss,ax       ;now establish our stack frame
    mov sp,bx
    mov bp,bx       ;bp = sp  this is critical for C
```

(continued on next page)

Listing One *(Listing continued, text begins on page 44.)*

```
    sti                 ;ok for an interrupt to occur
            ;now we have set up the environment for C
            ;STEP 4
            ; Call our first C function,
            ; passing the command code from the request header as
            ; a parameter.  (void)driver_functions(cmd_code);

    xor ax,ax                   ;clear out ax
    mov bx,offset driver_rh_    ;bx points to our request header
    mov al,byte ptr [bx + 2]    ;3rd byte of request header is command code
    push ax                     ;to pass parameter(s) to a C function,
    call driver_functions_      ;  push it(them) on stack and call function
    pop ax                      ;caller must remove parameter(s) from stack


            ;STEP 5
            ; At this point we are done with the function our driver
            ; was to perform.  Now we must copy our private request header
            ; back into the original request header DOS gave us a pointer
            ; to back in the strategy function.

    mov es,cs:req_hdr_seg       ;load address for orig RH into es:di
    mov di,cs:req_hdr_off
    mov si,offset driver_rh_    ;ds:si  our (updated) copy
    cld                         ;set direction to increment
    xor cx,cx                   ;clear out cx
    mov cl,[si]                 ;length is at first byte
    rep movsb                   ; ... and copy it


            ;STEP 6
            ; Restore machine state saved in STEP 0
            ; and return to DOS

    cli                     ;machine state restore should not be interrupted
    mov ax,cs:caller_ss     ;switch to caller's stack frame
    mov ss,ax
    mov ax,cs:caller_sp
    mov sp,ax
    popf                    ; ... pop all of it ...
    pop bp
    pop di
    pop si
    pop dx
    pop cx
    pop bx
    pop ax
    pop es
    pop ds
    sti                     ;enable interrrupts
    ret                     ; ... back to dos and bye bye

prn_driver    endp          ;end of driver code


codeseg    ends



    ;Data segment, contains our local stack space
    ;  and C addressable copy of request header

dataseg  segment para public 'DATA'

    public c_stack_top

    db  PRIVATE_STACK_SIZ dup (0)
c_stack_top  label  word

    public  driver_rh_
driver_rh_  db  32 dup (0)    ;C addressable copy of request header
dataseg    ends
END
```

End Listing One

Listing Two

```
/**  rh.h
     Device Driver Request Header structure definition
     and status word #define's
copyright (c) 1987 Andy Klein
**/


typedef struct
{
     unsigned char length,        /* length of header */
                   unit,          /* unit code ... which unit to use */
                   cmd;           /* command to execute */
     unsigned int  status;        /* status of operation */
     unsigned char reserved[8],
                   media_type;    /* media descriptor byte, block dev only */
     unsigned int  xfer_buf_offset,
                   xfer_buf_segment,
                   xfer_count;
     char          dummy[32 - 20];
                                  /* data for operation */
} request_hdr;


#define  ERROR_MASK          32768
#define  BUSY_MASK           1024
#define  DONE_MASK           512

#define  WRITE_PROTECTED     0x00
#define  UNKNOWN_UNIT        0x01
#define  DEV_NOT_READY       0x02
#define  UNKNOWN_CMD         0x03
#define  CRC_ERROR           0x04
#define  BAD_DRIVE_REQ_LEN   0x05
#define  SEEK_ERROR          0x06
```

(continued on next page)

Listing Two *(Listing continued, text begins on page 44.)*

```
#define  UNKOWN_MEDIA        0x07
#define  SECTOR_NOT_FOUND    0x08
#define  PRN_NO_PAPER        0x09
#define  WRITE_FAULT         0x0A
#define  READ_FAULT          0x0B
#define  GENERAL_FAILURE     0x0C
#define  INVALID_DISK_CHG    0x0F
```

End Listing Two

Listing Three

```
/**  drvfunc.c
        Device Driver for DOS 2.x, 3.x
        This is the function table that is called once
        the interrupt function has established addressability
copyright (c) 1987 Andy Klein
**/
#define LAST_FUNCTION 15
#define Q_FUNCTIONS   LAST_FUNCTION + 1

extern void init(), output_status(), output(), output_flush(),
    output_verf(), bad_cmd();

    /**
        function_table is an array of pointers to functions
        returning nothing (void).  It is analogous to a jump table
        in assembler.
    **/
void (* function_table[Q_FUNCTIONS])() = {
    /* 0 */        init,
    /* 1  */    bad_cmd,            /* media check */
    /* 2  */    bad_cmd,            /* build bpb */
    /* 3  */    bad_cmd,            /* ioctl input */
    /* 4  */    bad_cmd,            /* input */
    /* 5  */    bad_cmd,            /* input no wait */
    /* 6  */    bad_cmd,            /* input status */
    /* 7  */    bad_cmd,            /* input_flush */
    /* 8  */    output,
    /* 9  */    output_verf,
    /* 10 */    output_status,
    /* 11 */    output_flush,
    /* 12 */    bad_cmd,            /* ioctl output */
    /* 13 */    bad_cmd,            /* open device */
    /* 14 */    bad_cmd,            /* close device */
    /* 15 */    bad_cmd             /* removable_media */
};


void
driver_functions(cmd)
int cmd;
{
    if ( cmd > LAST_FUNCTION )
        (void) bad_cmd();
    else
        (void) (* function_table[cmd])();
}/* driver_functions() */
```

End Listing Three

Listing Four

```
/**  error.c
     Error handler for printer driver
copyright (c) 1987 Andy Klein
**/
#include "rh.h"

extern request_hdr driver_rh;

void
bad_cmd()
{
    driver_rh.status = ERROR_MASK | UNKNOWN_CMD;
```

(continued on page 74)

Listing Four *(Listing continued, text begins on page 44.)*

```
}


#define TIME_OUT  1
#define IO_ERR    8
#define NO_PAPER  32
#define BUSY      128


void
driver_error(stat)
int stat;
{
    int err_code;

    if ( stat & ERROR_MASK )
        stat ^= ERROR_MASK;
    switch ( stat )
    {
        case TIME_OUT:    err_code = DEV_NOT_READY;
                          break;
        case IO_ERR:      err_code = GENERAL_FAILURE;
                          break;
        case NO_PAPER:    err_code = PRN_NO_PAPER;
                          break;
        case BUSY:        err_code = DEV_NOT_READY;
                          break;
        default:          err_code = GENERAL_FAILURE;
                          break;
    }
    driver_rh.status = ERROR_MASK | err_code;
}/* driver_error() */
```

End Listing Four

Listing Five

```
/**  init.c
       Printer driver initialization
copyright (c) 1987 Andy Klein
**/
#include "rh.h"

extern request_hdr driver_rh;

char *title = "\nPrinter Device Driver for Okidata 92";
char *copyw  = "copyright 1987 (c) Andy Klein\n";


void
init()
{
    extern unsigned _Uend;
              /* _Uend is inserted by the Aztec linker */
    unsigned show_cs();
    void reset_printer(), initialize_oki92();

    puts(title);
    puts(copyw);
    (void) reset_printer();
    (void) initialize_oki92();
         /* set ending address of driver, set status word */
    driver_rh.xfer_buf_segment = show_cs();
    driver_rh.xfer_buf_offset = (unsigned int)&_Uend;
    driver_rh.status = DONE_MASK;
} /* init() */


#define ELITE_FONT  28

void
initialize_oki92()
{
    char_2_prn(ELITE_FONT);
}/* initialize_oki92() */
```

End Listing Five

Listing Six

```
;  show_cs.asm
;     c call is (unsigned) show_cs();
;     Returns contents of cs register
;copyright (c) 1987 Andy Klein

include lmacros.h

procdef         show_cs
    mov ax, cs
    pret
    pend show_cs

finish
```

End Listing Six

Listing Seven

```
/**  output.c
        Printer driver output functions
copyright (c) 1987 Andy Klein
**/
#include "rh.h"

extern request_hdr driver_rh;

void
output()
{
    int stat;
    unsigned  xfer_off, bytes_xferd;
    register char outch;
    char fetch_char();

    xfer_off = driver_rh.xfer_buf_offset;
    for ( bytes_xferd = 0; bytes_xferd < driver_rh.xfer_count;
            ++bytes_xferd, ++xfer_off )
    {
        outch = fetch_char(driver_rh.xfer_buf_segment, xfer_off);
        if ( (stat = char_2_prn(outch)) )
        {
            (void) driver_error(stat);
            break;
        }
    }

    driver_rh.xfer_count = bytes_xferd;
    if ( ! stat )
        driver_rh.status = DONE_MASK;
    else
        driver_rh.status = stat;
}/* output() */


void
output_verf()
{
    (void) output();
}/* out_verf() */



void
output_status()
{
    /**
        if device is currently doing an operation,
        driver_rh.status = BUSY_MASK & DONE_MASK;
        else if a write can begin immediately,
        driver_rh.status = 0 & DONE_MASK;
    **/
    if ( printer_busy() )
        driver_rh.status = BUSY_MASK & DONE_MASK;
    else
```

(continued on next page)

Listing Seven *(Listing continued, text begins on page 44.)*

```
        driver_rh.status = DONE_MASK;
}/* output_status() */


void
output_flush()
{
    /**
        terminate all pending requests,
        meaningless for this device
    **/
    driver_rh.status = DONE_MASK;
}/* output_flush() */
```

End Listing Seven

Listing Eight

```
;  prlport.asm
;    low level parallel port output functions
;copyright (c) Andy Klein 1987


include lmacros.h       ;Aztec C macros for assembly language functions
                        ; for other environments replace with calling
                        ; conventions for your complier


ERROR_MASK    equ    32768
PRINTER_INT   equu    017H
PRINTER_ID    equ    0            ;lpt1 = 0, lpt2 = 1, lpt3 = 2

    ;printer commands, put into ah
```

```
PRINT_CHAR    equ     0
INIT_PRN      equ     1
READ_STAT     equ     2

    ;printer status byte error codes, returned in ah
TIME_OUT      equ     1
IO_ERR        equ     8
NO_PAPER      equ     32
BUSY          equ     128

PRINTER_FAILURE    equ    TIME_OUT+IO_ERR+NO_PAPER


    ;C call is (void) reset_printer();
    ;no value returned
procdef reset_printer
        mov ah,INIT_PRN
        mov dx,PRINTER_ID
        sti
        int PRINTER_INT
        xor ax,ax
        pret
pend reset_printer


    ;C call is printer_busy();
    ;returns 0 (FALSE) if not busy, nonzero (TRUE) if busy
procdef printer_busy
        mov ah,READ_STAT
        mov dx,PRINTER_ID
        sti
        int PRINTER_INT
        and ah,BUSY
        jz  printer_is_busy
```

(continued on next page)

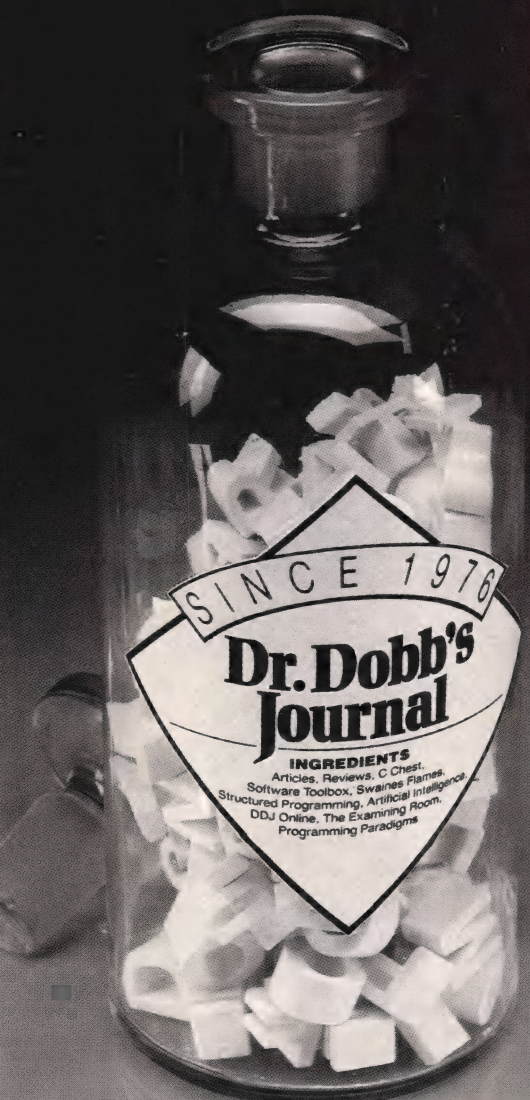
Listing Eight *(Listing continued, text begins on page 44.)*

```
        xor ax,ax
        pret
    printer_is_busy:
        mov ax,1
        pret
pend printer_busy



    ;  C call is    (char) fetch_char(segm, offs);
    ;     gets the character at segm:offs and
    ;     returns it

procdef fetch_char,<<segm,word>,<offs,word>>
        push ds
        mov bx,offs
        mov ds,segm
        mov al,byte ptr ds:[bx]
        and ax,0ffH
        pop ds
        pret
pend fetch_char


    ;C call is char_2_prn(outch);
    ;    prints outch on PRINTER_ID
    ; returns 0 if ok, error code

procdef char_2_prn,<<outch,byte>>
        mov al,outch
        mov ah,PRINT_CHAR
        mov dx,PRINTER_ID
        sti
```

```
        int PRINTER_INT
        test ah,PRINTER_FAILURE
        jnz printer_has_failed
        xor ax,ax
        pret
    printer_has_failed:
        mov al,ah                ;result code in ah, mov it to al
        and ax,PRINTER_FAILURE
        pret
pend char_2_prn


finish
```

End Listings

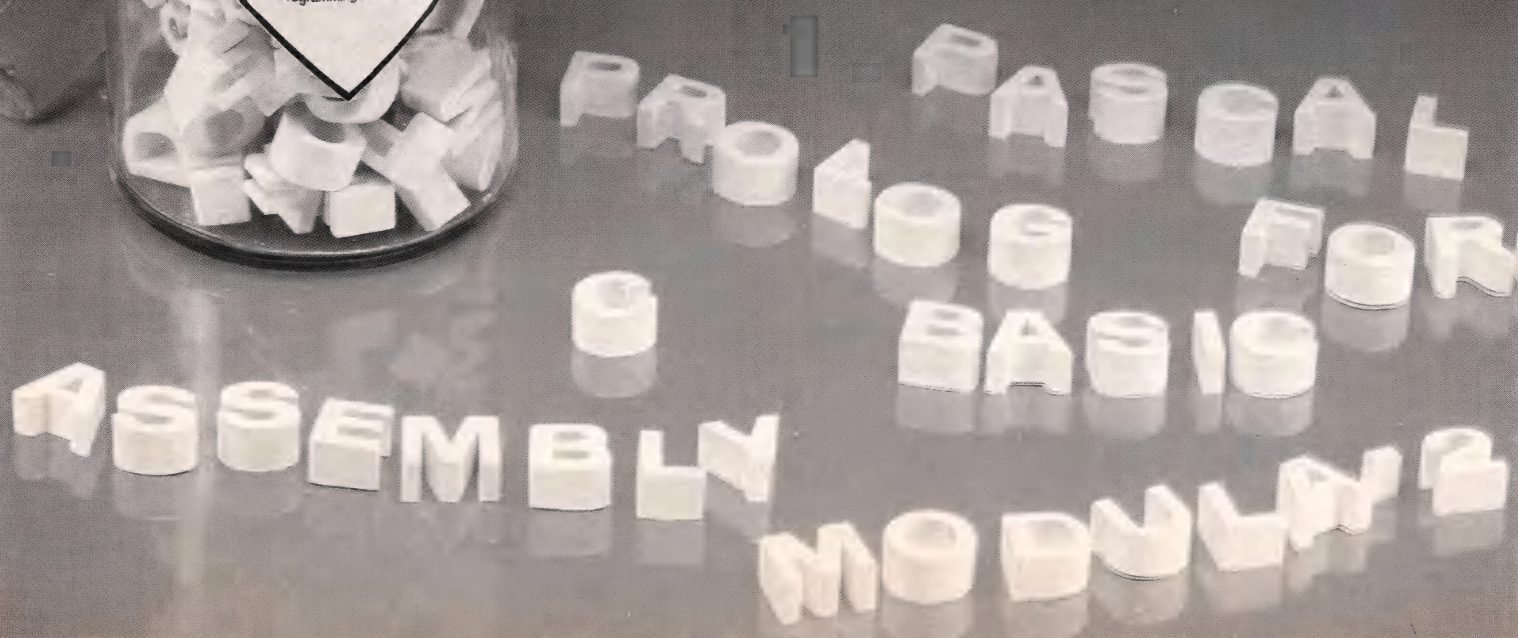
THE CURE FOR COMMON CODE



Are you getting the recommended monthly allowance of C, Assembly, Forth, Pascal, BASIC or Modula-2? Subscribe to *Dr. Dobb's Journal of Software Tools* and you won't catch any nasty bugs again!

Each month the Doctor brings you aid for ailing algorithms and the cure for common code. For the latest developments in software design and pages of code that will make you a more productive programmer, take the Dr. Dobb's prescription.

For more than a decade, the programming elite have known *Dr. Dobb's Journal* to be the foremost source of software tools. Subscribe now and get your monthly dose from the Doctor.



COMMENTS & SUGGESTIONS

Dear Reader, **September 1987, #131**

Dr. Dobb's has a long tradition of listening to its readers. We like to hear when something really helps or, for that matter, bothers you. In this hectic world of ours, however, it is often difficult to take time to write a letter. This card provides you with an easy way to correspond and, if you include your name and address, we may use appropriate comments in The Letters column. Simply fill it out and drop it in the mail. —Ed

Which articles or departments did you enjoy the most this month? Why?

_____

_____

_____

Comments or suggestions _____

_____

_____

_____

Name: _____

Address: _____

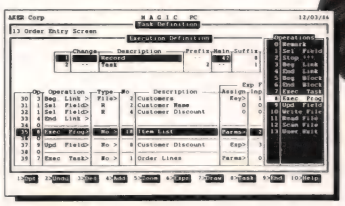
_____

Make $500/hr.

Now develop DBMS applications 10 times faster for only $199 with MAGIC PC — or your money back!

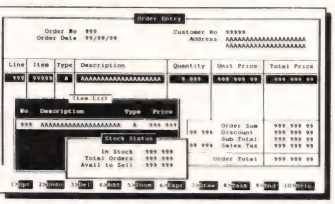
Database programmers, why waste your time hacking out code?
Imagine how much faster and more profitable you'd be if you could whip up powerful database applications without the time-consuming coding pains...
Introducing Magic PC from Aker, your professional dream come true. It's not another line-by-line syntax treadmill like any DBMS or 4GL.
Finally you can program as quickly as you design, while you delegate all the mundane and redundant coding tasks to Magic PC.

Program 10 times faster



Develop relational database applications 10 times faster using a visual design-driven interface. Instead of writing mountains of "how to" procedural code, you quickly place your program design specs in Execution Tables and Magic PC's engine executes them automatically. Don't lose any more time editing and debugging programs by hand.

Incredible Zoom power



Magic PC's phenomenal Zoom power magically co-executes related programs through nested Zoom windows smoothly with auto data scrolling in all directions. While Zooming, query and transfer data across windows or even Zoom deeper.

No more maintenance!

Change your programs on the fly without any manual maintenance responsibility. Magic PC automatically updates your changes online since all the data describing your design (data dictionary, programs and menus) make up a single file, self-maintaining Integrated Library.

Magic PC does it all

Design your entire database application with only one comprehensive development system. Generate both online programs (screens, windows, menus), as well as batch programs (reports, updates, import/export, etc.) with full color and graphics. You no longer fall between the cracks dealing with separate and inconsistent programming utilities.

Free LAN features

Develop multi-user applications for local area networks with Magic PC's automatic support for file and record locking security.

Quick prototyping

Prototype a complete working application in just hours and get immediate customer feedback to finalize the design. It's a true time-saver.

Stand-alone runtime

Distribute your applications and protect your design with a low cost runtime engine. It has the friendliest end-user visual interface you've ever seen with built-in, menu-driven and syntax-free data retrieval power.

Jeff Duntemann, PC Tech Journal:

"*Magic PC is probably the best integrated database application generator that we have seen... very smooth system, and smoothness comes at a premium these days.*" Also recommended by PC Magazine, PC World, PC Week, Computer Language, Data Based Advisor and many more around the world.

Try it for $19.95

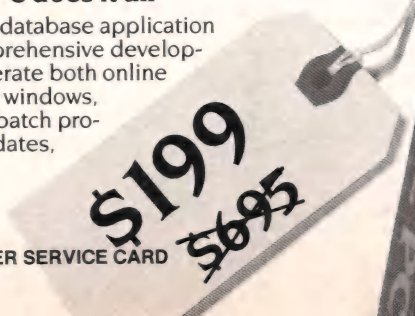
If you develop database applications for a living, you can't afford not to try Magic PC for yourself right now. For $19.95 you'll get the Magic PC Tutorial software and documentation for hands-on evaluation, complete with a step-by-step guide to develop an Order Entry sample application in just a few hours.

Magic PC ~~$695~~ $199

No kidding! For a limited time only, save almost $500 off the $695 list price, and get the complete unprotected Magic PC software for only $199 at our special introductory non-resale price.

Money back guarantee

Even at $199 you can't go wrong with our no-risk guarantee: keep it only if it makes magic for you, or we'll buy it back within 30 days less $19.95 restocking fee.



Call this toll-free number now with your credit card or COD charge. Dealer pricing available.

800-345-MAGIC • in CA 714-250-1718

Or send this order coupon with your payment to:
Aker 18007 Skypark Cir. B2, Irvine, CA 92714

___Magic PC at ~~$695~~ $199 (add $10 P&H)
___Magic PC Tutorial at $19.95 (add $5 P&H)
In CA add applicable tax

1 2 3 4 5 6 7 8 9 0

Name _____
Company _____
Address (no POB) _____
City/St/Zip _____
Phone _____
Check enclosed, or charge: VISA MasterCard AMERICAN EXPRESS
Acct. No. _____
Exp. Date ___/___
Name on card _____
Signature _____

System Requirements: IBM PC, XT, AT, PS/2 and 100% compatible, PC-DOS 2.0 or later, 512K, hard disk. All trademarks acknowledged.

MAGIC PC
The Visual Database Language
by AKER

CIRCLE 369 ON READER SERVICE CARD

C CHEST

Listing One *(Text begins on page 106.)*

```
 1| #define PUBLIC
 2|
 3| #ifdef  DEBUG
 4| #       define PRIVATE
 5| #       define D(x) x
 6| #else
 7| #       define PRIVATE static
 8| #       define D(x)
 9| #endif
10|
11| #ifdef MSDOS
12| #       define MS(x) x
13| #       define UX(x)
14| #else
15| #       define MS(x)
16| #       define UX(x) x
17| #endif
```

End Listing One

Listing Two

```
 1| /* Defines for IBM-PC Hardware */
 2|
 3| /* Timer defines:
 4|  *
 5|  * TIMR_CLK       The number of system clock cycles in a second
 6|  *                (ie. the input frequency of the counter/timer).
 7|  * TIMR_TICKS(d)  The number of ticks needed to get a delay of d
 8|  *                seconds. 'd' can be a fraction: TIMR_TICKS(.5).
 9|  *
10|  * TIMR_CTRL      Address of control port
11|  * TIMR_2_DATA    Address of counter 2 data port
12|  * TIMR_2_LOAD    Control word to load new count into
13|  *                timer 2 (the speaker--2 bytes, lsb first).
14|  *                Timer initilized in mode 3 (square wave)
15|  * TIMR_0_LOAD    Same but for timer 0 (system clock).
16|  * TIMR_0_DATA
17|  */
18|
19| #define TIMR_CLK         1193180L
20| #define TIMR_TICKS(d)    (int)((double)(d) * (TIMR_CLK/65536.0))
21|
22| #define TIMR_CTRL        0x43
23|
24| #define TIMR_0_DATA      0x40
25| #define TIMR_0_LOAD      0x36
26|
27| #define TIMR_2_DATA      0x42
28| #define TIMR_2_LOAD      0xb6
29|
30| /* Programmable peripheral interface:
31|  *
32|  *      PPI              Base address of interface
33|  *      PPI_SPKR         Bit mask to enable speaker (bit 0 is
34|  *                       gate on timer chip, bit 1 actually
35|  *                       enables the speaker).
36|  */
37|
38| #define PPI              0x61
39| #define PPI_SPKR         0x03
40|
41| /*---------------------------------------------------------------
42|  * Make the speaker beep at a particular frequency:
43|  *
44|  * SETFRQ(freq);     Sets the frequency, freq can be floating
45|  *                                                        point
46|  * SPKR_ON();        Turns the speaker on.
47|  * SPKR_OFF();       Turns it off again.
48|  */
49|
50| #define SETFRQ( freq )
51|     if( 1 )
52|     {
53|         unsigned int count;
```

```
54|
55|         count = TIMR_CLK / freq ;
56|
57|         outp( TIMR_CTRL  , TIMR_2_LOAD          );
58|         outp( TIMR_2_DATA,  count       & 0xff  );
59|         outp( TIMR_2_DATA, (count >> 8) & 0xff  );
60|     }
61|     else
62|
63| #define SPKR_ON()  outp( PPI, inp(PPI) |  PPI_SPKR )
64| #define SPKR_OFF() outp( PPI, inp(PPI) & ~PPI_SPKR )
```

End Listing Two

Listing Three

```
 1| /* These #defines are the frequencies 12 notes of the octave
 2|  * starting with middle C. Multiply by two to go up an octave,
 3|  * divide by two to go down. This is an equal-tempered scale
 4|  * so each note is derrived by multiplying the previous note
 5|  * by the twelfth root of two. Note that there's a little
 6|  * round-off error here but this error isn't audible.
 7|  */
 8|
 9| #define TWELFTH_ROOT_OF_TWO     1.059463095
10|
11| #define C4          (261.6256)          /*      C  4    */
12| #define C4_SHARP    (277.1826)          /*      C# 4    */
13| #define D4          (293.6648)          /*      D  4    */
14| #define D4_SHARP    (311.1270)          /*      D# 4    */
15| #define E4          (329.6276)          /*      E  4    */
16| #define F4          (349.2282)          /*      F  4    */
17| #define F4_SHARP    (369.9944)          /*      F# 4    */
18| #define G4          (391.9954)          /*      G  4    */
19| #define G4_SHARP    (415.3047)          /*      G# 4    */
20| #define A4          (440.0000)          /*      A  4    */
21| #define A4_SHARP    (466.1638)          /*      A# 4    */
22| #define B4          (493.8833)          /*      B  4    */
```

End Listing Three

Listing Four

```
 1| #include <stdio.h>
 2| #include <tools/hardware.h>
 3|
 4| beep( freq, duration )
 5| double  freq;
 6| double  duration;
 7| {
 8|         /* Beep the bell on the IBM-PC for the indicated time
 9|          * (which may be fractional) at the indicated frequency.
10|          * Frequencies for various notes in an equal-tempered
11|          * scale are in <tools/notes.h>.
12|          */
13|
14|         SETFRQ( freq );
15|
16|         SPKR_ON();
17|
18|         delay( (double)duration );
19|
20|         SPKR_OFF();
21| }
22|
23| /*-----------------------------------------------------------*/
24| #ifdef MAIN
25| #include <tools/notes.h>
26|
27| main( argc , argv )
28| char    **argv;
29| {
30|         double atof();
31|
32| #       if 0
33|             beep(  C * 4 ,      atof(argv[1]) );
34| #       endif
35|
```

(continued on next page)

Listing Four

(Listing continued, text begins on page 106.)

```
36|         beep(  C ,      0.5 );
37|         beep(  D ,      0.5 );
38|         beep(  E ,      0.5 );
39|         beep(  F ,      0.5 );
40|         beep(  G ,      0.5 );
41|         beep(  A ,      0.5 );
42|         beep(  B ,      0.5 );
43|         beep(  C * 2 ,  0.5 );
44| }
45| #endif
```

End Listing Four

Listing Five

```
 1| #include <tools/hardware.h>
 2| #include <dos.h>
 3|
 4| delay( duration )
 5| double  duration;
 6| {
 7|         /*   Delay for the indicated number of seconds (may be
 8|          *   fractional.
 9|          */
10|
11|         unsigned long  start, elapsed ;
12|
13|         unsigned long i, t;
14|
15|         elapsed = TIMR_TICKS( duration );
16|
17|         for( start = ticks(); ticks() - start < elapsed ;)
18|                 ;
19| }
20|
21| ticks()
22| {
23|     /* Return the number of BIOS clock ticks since midnight.
24|      * The routine rolls over succesfully at midnight (1573040
25|      * is the number of clock ticks in a day and AL is zero
26|      * if the timer has not passed midnight since the last
27|      * call).
28|      */
29|
30|     union   REGS    regs;
31|
32|     regs.h.ah = 0;
33|
34|     int86( 0x1a, &regs, &regs );    /* Time-of-day interrupt */
35|
36|     return    ( regs.h.al ? 1573040L : 0 )
37|             + ( regs.x.cx << 16          )
38|             +   regs.x.dx
39|             ;
40| }
41|
42| #ifdef MAIN
43| main()
44| {
45|         printf("Should be five seconds between beeps\n\007");
46|         delay( 5.0 );
47|         printf("\007");
48| }
49| #endif
```

End Listing Five

Listing Six

```
 1|    PAGE   56,132
 2|    TITLE  SPEEDUP.ASM: System-clock-modification routines
 3| ;--------------------------------------------------------------
 4| DEBUG   equ 1   ; Set to 1 to make internal symbols public
 5| ;--------------------------------------------------------------
 6|
```

```
 7| _TEXT   SEGMENT  BYTE PUBLIC 'CODE'
 8| _TEXT   ENDS
 9| _DATA   SEGMENT  WORD PUBLIC 'DATA'
10| _DATA   ENDS
11| CONST   SEGMENT  WORD PUBLIC 'CONST'
12| CONST   ENDS
13| _BSS    SEGMENT  WORD PUBLIC 'BSS'
14| _BSS    ENDS
15| DGROUP  GROUP   CONST,  _BSS,   _DATA
16|         ASSUME  CS: _TEXT, DS: DGROUP, SS: DGROUP, ES: DGROUP
17|
18| EXTRN   __chkstk:NEAR
19|
20| ;--------------------------------------------------------
21|
22| TIMR_CTRL   = 43H       ; address of timer control port
23| TIMR_0_DATA = 40H       ; address of counter 0 data port
24| TIMR_0_LOAD = 36H       ; control word for timer
25|
26| ;--------------------------------------------------------
27|
28| _TEXT     SEGMENT
29|
30| ;--------------------------------------------------------
31| ; Misc. variables. Note that I'm putting all these in the
32| ; code (_TEXT) segment so that I can find them when an
33| ; interrupt comes along. The PUBLIC statements are just for
34| ; debugging.
35|
36| old_int    equ  $
37| old_off    dw   ?       ; Offset of old timer interrupt
38|                         ;  service routine.
39| old_seg    dw   ?       ; Segment address of same.
40|
41| service    dw   ?       ; User-supplied interrupt service
42|                         ; routine (offset).
43|
44| old_ds     dw   ?       ; segments for running program.
45| tick_reset dw   ?
46| numticks   dw   ?        ; Initialized to tick_reset, decre-
47|                         ; mented on each timer interrupt,
48|                         ; reset to the speedup factor (and the
49|                         ; old service routine is called) when
50|                         ; it reaches zero.
51|
52| stack       dw  64 dup (0) ; Local stack for service routine
53|                            ; 18 bytes are used by real service
54|                            ; routine, the rest is available
55|                            ; for the user service routine.
56| stack_end   dw  ?
57| old_sp      dw  ?
58| old_ss      dw  ?
59|
60| IF DEBUG
61|     PUBLIC  old_int,  old_off,    old_seg,  old_ds
62|     PUBLIC  service,  tick_reset, numticks, serv
63| ENDIF
64|
65| ;--------------------------------------------------------
66| ; cli();  sti();  Disable and enable interrupts.
67| ;
68| PUBLIC  _cli, _sti
69|
70| _cli    PROC NEAR
71|         cli
72|         ret
73| _cli    ENDP
74|
75| _sti    PROC NEAR
76|         sti
77|         ret
78| _sti    ENDP
79|
```

(continued on next page)

Listing Six *(Listing continued, text begins on page 106.)*

```
80| ;------------------------------------------------------------
81| ; speedup( factor, routine )
82| ; int factor, (*routine)();
83| ;
84| ;  Speed up the system clock by the indicated factor.
85| ;  Call the indicated subroutine on every timer interrupt.
86| ;  and call the default clock routine as well every "factor"
87| ;  ticks.
88| ;
89| ; Offsets to arguments:
90| ;       factor  = [bp+4]
91| ;       routine = [bp+6]
92| ;
93| PUBLIC  _speedup
94|
95| _speedup        PROC NEAR
96|         push    bp
97|         mov     bp,sp
98|         xor     ax,ax
99|         call    __chkstk
100|
101|        mov     ax,[bp+6]           ; service = offset of new
102|        mov     _TEXT:service,ax    ;  routine.
103|        mov     _TEXT:old_ds,ds     ; remember current DS too.
104|        mov     ax,[bp+4]           ; tick_reset = numticks
105|        mov     _TEXT:tick_reset,ax ; = ax = factor
106|        mov     _TEXT:numticks,ax   ;
107|
108|        mov     al,TIMR_0_LOAD      ; Set up timer for load
109|        out     TIMR_CTRL,al        ;
110|        mov     ax,[bp+4]           ; if( factor == 1 )
111|        cmp     ax,01H              ; {
112|        jne     do_div              ;   use 0 for the ouput count
113|        mov     ax,0                ; }
114|        jmp     load                ; else
115| do_div:                            ; {
116|        mov     ax,00000H           ;     Number of ticks =
117|        mov     dx,00001H           ;           65536/factor
118|        mov     bx,[bp+4]           ;     BX = factor.
119|        div     bx                  ;    AX = number of ticks
120| load:                              ; }
121|        out     TIMR_0_DATA,al      ; Send new count to timer
122|        mov     al,ah               ;
123|        out     TIMR_0_DATA,al      ;
124|
125|                                    ; Get the old vector
126|        mov     ah,35H              ;
127|        mov     al,08H              ;
128|        int     21H                 ;
129|        mov     _TEXT:old_off,bx    ;
130|        mov     _TEXT:old_seg,es    ;
131|
132|                                    ; set up the new vector
133|        mov     ah,25H              ;
134|        mov     al,08H              ;
135|        mov     dx,OFFSET _TEXT: serv ;
136|        push    ds                  ;
137|        push    cs                  ;
138|        pop     ds                  ;
139|        int     21H                 ;
140|        pop     ds                  ;
141|
142|        mov     sp,bp
143|        pop     bp
144|        ret
145|
146| _speedup        ENDP
147|
148| ;------------------------------------------------------------
149| ; Actual interrupt service routine. This routine saves the
150| ; environment, calles the user-supplied C service routine,
151| ; and then calls the default service routine if necessary.
```

```
152| ; The service routine runs under its own stack so stack
153| ; checking should be disabled with either the /Gs command-
154| ; line switch or the "#pragma check_stack[+|-]" directive.
155| ;
156| serv    PROC    NEAR
157|
158|         push    ax                          ; Save AX on old stack
159|
160|         mov     _TEXT:old_sp,sp             ; Set up local
161|         mov     _TEXT:old_ss,ss             ; stack
162|         push    cs                          ;
163|         pop     ss                          ;
164|         mov     sp,offset _TEXT: stack_end ;
165|
166|         push    bx              ; Set up C environment:
167|         push    cx              ;       save everything (the
168|         push    dx              ;       flags are saved as
169|         push    bp              ;       part of the interrupt
170|         push    si              ;       processing).
171|         push    di              ;
172|         push    ds              ;
173|         push    es              ;
174|                                  ;
175|         mov     ds,_TEXT:old_ds ;       fix the data segment
176|         mov     es,_TEXT:old_ds ;       and extra segment
177|
178|         cli
179|         call    word ptr _TEXT:service  ; Call C subroutine
180|         sti
181|                                         ;
182|         pop     es                      ; restore everything
183|         pop     ds                      ; but AX
184|         pop     di                      ;
185|         pop     si                      ;
186|         pop     bp                      ;
187|         pop     dx                      ;
188|         pop     cx                      ;
189|         pop     bx                      ;
190|
191|         mov     ss,_TEXT:old_ss         ; Restore original
192|         mov     sp,_TEXT:old_sp         ;       stack.
193|                                         ;
194|         dec     _TEXT:numticks          ; if(--numticks > 0)
195|         jle     do_old_int              ; {
196|         mov     al,20h                  ;     send EOI
197|         out     20h,al                  ;
198|         pop     ax                      ;     restore ax
199|         iret                            ; }
200|                                         ; else
201| do_old_int:                             ; {
202|         mov     ax,_TEXT:tick_reset     ;     numticks
203|         mov     _TEXT:numticks,ax       ;         = tick_reset;
204|         pop     ax                      ;     restore ax
205|         jmp     dword ptr _TEXT:old_int ;     jmp to old vector
206|                                         ; }
207| serv    ENDP
208|
209| ;-------------------------------------------------------------
210|
211| PUBLIC  _slowdown
212|
213| _slowdown       PROC NEAR
214|
215|         push    bp
216|         mov     bp,sp
217|         xor     ax,ax
218|         call    __chkstk
219|
220|         mov     ax,_TEXT:old_off ; See if the interrupts have
221|         or      ax,ax           ;                   changed.
222|         jz      no_int          ; No, don't fix them then
223|
224|                                 ; restore old timer interrupt
225|         push    ds              ;
226|         mov     ah,25H          ;
```

(continued on next page)

C CHEST

Listing Six

(Listing continued, text begins on page 106.)

```
227|        mov     al,08H          ;
228|        mov     ds,_TEXT:old_seg ;
229|        mov     dx,_TEXT:old_off ;
230|        int     21H             ;
231|        pop     ds              ;
232|
233| no_int:
234|        mov  al,TIMR_0_LOAD     ; Restore default system
235|        out  TIMR_CTRL,al       ; clock tick rate
236|        mov  al,0
237|        out  TIMR_0_DATA,al
238|        out  TIMR_0_DATA,al
239|
240|        mov     sp,bp
241|        pop     bp
242|        ret
243|
244| _slowdown       ENDP
245|
246| _TEXT   ENDS
247| END
```

End Listing Six

Listing Seven

```
 1| #include <stdio.h>
 2| #include <signal.h>
 3| #include <stdarg.h>
 4| #include <ctype.h>
 5| #include <tools/notes.h>        /* Frequencies of notes */
 6| #include <tools/hardware.h>     /* TIMR_CLK             */
 7| #include <tools/debug.h>        /* D() macro            */
 8|
 9| /* CLICK.C      A polyrhythmic metronome. Usage is described
10|  *              in the usage_msg[], below.
11|  *
12|  *              (c) 1987, Allen I. Holub. All rights reserved.
13|  *-------------------------------------------------------------
14|  */
15|
16| extern double   ceil ( double );
17| extern double   floor( double );
18|
19| /*-------------------------------------------------------------
20|  * The compiler truncates floating point numbers when they're
21|  * converted to int. This macro rounds as it converts.
22|  */
23|
24| #define ROUND(x) ((int)( ((x) > 0.5) ? ceil ((double)(x)) \
25|                                      : floor((double)(x)) )  )
26|
27| /*-------------------------------------------------------------
28|  * TICKS(x)  converts a metronome count to clock ticks.
29|  *
30|  * With a speedup factor 4, a tick happens 72.84 times/second
31|  * (every .01373 seconds, more or less). A speedup factor of
32|  * 2 yields half this number: 36.42 times/sec, or an interrupt
33|  * every .02746 seconds).
34|  *
35|  * A metronome 60 is 1 Hz, 120 is 2 Hz, etc.
36|  *
37|  * seconds         ==    60 / metronome_count
38|  * ticks in second == (  60 / metronome_count ) * ONE_TICK
39|  *                 == ( (60 * ONE_TICK) / metronome_count )
40|  */
41|
42| #define FACTOR          16                  /* Speedup factor */
43| #define DEFAULT_TICK    (TIMR_CLK / 65536.0) /* ticks / second */
44| #define ONE_TICK        (DEFAULT_TICK * FACTOR)
45|
46| #define TICKS(x)        ROUND( (60.0 * ONE_TICK) / (x) )
47|
```

```
48| /*-------------------------------------------------------------*/
49|
50| #define min(a,b)        ((a) < (b) ? (a) : (b))
51| #define max(a,b)        ((a) > (b) ? (a) : (b))
52|
53| #define MAX_MEASURES    1280    /* Max # of measures/track   */
54| #define NUM_TRACKS      4       /* Number of tracks          */
55|
56| #define WARNING         (NUM_TRACKS + 1)
57|
58| /*-------------------------------------------------------------*/
59|
60| typedef unsigned char   uchar;
61| typedef unsigned int    uint;
62|
63| typedef struct
64| {
65|     uchar num_beats;       /* # of beats remaining in measure */
66|     uchar remainder;        /* Number of ticks to get in synch */
67|     uint  cur_tick;         /* Current tick for this beat      */
68|     uint  ticks_per_beat ; /* # of clock ticks between beats   */
69|     uint  metronome : 14 ; /* metronome count in this measure  */
70|     uint  silent    : 1  ; /* this measure is silent           */
71|     uint  warning   : 1  ; /* warning tone at downbeat         */
72| }
73| MEASURE;
74|
75| typedef MEASURE TRACK [ MAX_MEASURES ];
76|
77| TRACK   Tape      [ NUM_TRACKS ];
78| MEASURE *Measure [ NUM_TRACKS ];  /* Current measure on */
79|                                   /* each track of Tape.*/
80| int     Lineno = 0;               /* Input line number  */
81|
82| int     Ring_bell = 0;  /* 0 if the bell shouldn't ring.
83|                          * Set to 1 for track 1, 2 for track
84|                          * 2, etc.
85|                          */
86|
87| int     Collision = 0 ; /* If two track collide, the track
88|                          * number of the second one is
89|                          * put here.
90|                          */
91|
92| int     Downbeat = 1;   /* Incremented by the interrupt
93|                          * service routine on every
94|                          * downbeat from track 0;
95|                          */
96|
97| int     Done = 0;       /* Set to 1 by interrupt service
98|                          * routine when it gets to the
99|                          * end of the tape.
100|                         */
101|
102| int     Numticks = 0;   /* For debugging, incremented on
103|                          * every timer interrupt.
104|                          */
105|
106| /*-------------------------------------------------------------*/
107|
108| char    *Usage_msg[] =
109| {
110|     "Usage: click [-d] inputfile",
111|     "",
112|     " -d (for dull) don't use different notes for different",
113|     "    tracks",
114|     "",
115|     "A polyrhythmic metronome. Four independent \"tracks\"",
116|     "are supported, with rhythms specified as follows:",
117|     "",
118|     "track 0: #4 @120 5/8, #6 @120 6/8, @120 4/4",
119|     "track 1: #4      6  , #6 @100 6/8",
120|     "",
```

(continued on next page)

Listing Seven *(Listing continued, text begins on page 106.)*

```
121|     "No line can be longer than 132 characters, but several",
122|     "track specifiers can be given. The basic notation is:",
123|     "\"[#N] [@Z] X[/Y],\" interpreted as N measures of X/Y at",
124|     "metronome Z. If the \"#N\" is missing, 1 is used. If",
125|     "the \"@Z\" is missing, the measure is stretched to",
126|     "synch with track one on the down beat. The \"/Y\" is",
127|     "optional. So, in the above example, the six beats in",
128|     "the first four measures of track 2 will synch up with",
129|     "track one, coming into synch on the downbeat of each",
130|     "measure. Each track may be up to 1000 measures long.",
131|     "Lines that don't begin with \"track\" are ignored",
132|     NULL
133| };
134|
135| /*-------------------------------------------------------------*/
136|
137| char    *skipwhite(p)
138| char    *p;
139| {
140|     /* Skip all characters that aren't part of a command */
141|
142|     while( *p && (isspace(*p) || *p == '\n') )
143|             p++ ;
144|
145|     return p;
146| }
147|
148| /*-------------------------------------------------------------*/
149|
150| err( fmt )
151| char    *fmt;
152| {
153|         /* Works like printf() but writes to standard error and
154|          * prints an input line number along with the message.
155|          */
156|
157|         va_list   args;
158|         va_start( args, fmt );
159|
160|         fprintf ( stderr, "line %d: ", Lineno );
161|         vfprintf( stderr, fmt, args );
162| }
163|
164| /*-------------------------------------------------------------*/
165|
166| init()
167| {
168|         /* Initialize The Measure array to point
169|          * at the first measure of each track on the Tape.
170|          */
171|
172|         register int i;
173|
174|         for( i = NUM_TRACKS; --i >= 0; )
175|                 Measure[i] = Tape[i] ;
176| }
177|
178| /*-------------------------------------------------------------*/
179|
180| print_tape( this_many )
181| {
182|     /* Print out the tape. If this_many is 0, only the
183|      * initialized measures are printed; otherwise, the
184|      * indicated number of measures are printed
185|      */
186|
187|     MEASURE      *p ;
188|     register int  i , measure_num ;
189|     int           amt;
190|
191|     for( i = 0; i < NUM_TRACKS ; i++ )
192|     {
193|         printf( "Track %d:\n", i );
```

```
194|        measure_num = 0;
195|        amt         = this_many;
196|
197|        for( p=Tape[i]; p->num_beats>0 || --amt >= 0; p++)
198|        {
199|            printf("    measure %2d: ", ++measure_num    );
200|            printf("[%2d ticks/beat " , p->ticks_per_beat );
201|            printf("+ %d], "          , p->remainder     );
202|            printf("cur_tick=%d, "    , p->cur_tick      );
203|            printf( "%d beats "       , p->num_beats     );
204|
205|            if( p->metronome )
206|                printf( "at %-5d ", p->metronome );
207|            else
208|                printf( "in synch " );
209|
210|            printf("%s", p->silent  ? "(mute)" : "" );
211|            printf("%s", p->warning ? "(warn)" : "" );
212|            printf("\n");
213|        }
214|    }
215| }
216|
217| /*-----------------------------------------------------------*/
218|
219| build_tracks( file_name )
220| char    *file_name;
221| {
222|     FILE        *fp;
223|     MEASURE     *mp;
224|     char        buf[133], *line;
225|     int         i;
226|     int         track;          /* Track number            */
227|     int         num_measures;   /* # of measures to repeat */
228|     int         metronome;      /* metronome count         */
229|     int         beats;          /* beats per measure       */
230|     int         ticks;          /* ticks per measure       */
231|     int         measure;        /* Current measure number  */
232|     int         silent;         /* measure is silent       */
233|     int         warning;        /* warning on last repeat  */
234|
235|     if( ! (fp = fopen(file_name,"r")) )
236|             return 0;
237|
238|     init();
239|
240|     while( line = fgets(buf,133,fp) )
241|     {
242|         ++Lineno;
243|
244|         line = skipwhite( line );
245|
246|         if( !(   line[0]=='t' && line[1]=='r'
247|               && line[2]=='a' && line[3]=='c' && line[4]=='k'
248|             )
249|         ) continue;
250|
251|         line += 5;                        /* Get track number   */
252|         line  = skipwhite (  line );
253|         track = stoi      ( &line );
254|         line  = skipwhite (  line );
255|
256|         if( *line == ',' || *line == ':' )
257|                 line++;
258|
259|         mp      = Measure   [ track ];  /* starting measure # */
260|         measure = mp - Tape [ track ];
261|
262|         while( *line )
263|         {
264|             num_measures = 1;
265|             metronome    = 0;
266|             silent       = 0;
267|             warning      = 0;
268|
```

(continued on next page)

Listing Seven *(Listing continued, text begins on page 106.)*

```
269|            for( line=skipwhite(line); *line; line=skipwhite(line) )
270|            {
271|                if( *line == ';' )              /* comment       */
272|                {
273|                        *line = 0;
274|                        break;
275|                }
276|                else if( *line == ',' || *line == ':' )
277|                {
278|                        ++line;                 /* end of measure */
279|                        break;
280|                }
281|                if( *line == '#' )              /* # of measures  */
282|                {
283|                    line         = skipwhite ( ++line );
284|                    num_measures = stoi      (  &line );
285|                }
286|                else if( *line == '@' )         /* metronome count */
287|                {
288|                    line      = skipwhite ( ++line );
289|                    metronome = stoi      (  &line );
290|                }
291|                else if( *line == 'w' || *line == 'W' )
292|                {
293|                    while( isalpha(*line ) )
294|                        ++line;
295|
296|                    warning = 1;
297|                }
298|                else if( isdigit( *line ) )     /* Time signature */
299|                {
300|                    if( (beats = stoi(&line)) == 0 )
301|                    {
302|                        err( "Illegal time signature\n" );
303|                        exit(1);
304|                    }
305|
306|                    if( *line == '/' )          /* Throw away bottom  */
307|                        ++line;                 /* of time signature. */
308|
309|                    while( isdigit( *line ) )
310|                        line++;
311|                }
312|                else if( *line == '(' || *line == ')' )
313|                {
314|                    ++ line   ;
315|                    silent = 1;
316|                }
317|                else
318|                    err("<%c> is illegal in measure spec.\n", *line );
319|            }
320|
321|            for(; --num_measures >= 0; mp++, measure++ )
322|            {
323|                if( metronome )
324|                {
325|                    mp->metronome = metronome;
326|                    ticks         = TICKS(metronome) * beats ;
327|                }
328|                else
329|                {
330|                    mp->metronome = 0;
331|                    ticks = Tape[0][measure].ticks_per_beat
332|                          * Tape[0][measure].num_beats
333|                          ;
334|                }
335|
336|                if( num_measures == 0 ) /* last in series */
337|                        mp->warning = warning;
338|
339|                mp->silent         = silent ;
340|                mp->num_beats      = beats  ;
341|                mp->cur_tick       = ticks / beats;
```

```
342|                mp->ticks_per_beat = ticks / beats;
343|                mp->remainder      = ticks % beats;
344|
345|                D( printf("loading track %d, ", track        );)
346|                D( printf("measure %d: "      , measure      );)
347|                D( printf("%d beats/measure " , beats        );)
348|                D( printf("at metronome %d\n" , mp->metronome);)
349|            }
350|
351|            Measure[ track ] = mp ;
352|        }
353|    }
354|
355|    init();
356|    D( print_tape( 0 ); )
357|    return 1;
358| }
359|
360| /*-------------------------------------------------------------*/
361|
362| #pragma check_stack-    /* Turn off stack probes. This pragma */
363|                         /* is Microsoft-compiler dependent.   */
364|
365| timer_intr()
366| {
367|    /* Interrupt service routine for timer interrupt */
368|
369|    MEASURE  **mp, *p ;
370|    int         i,  did_nothing ;
371|
372|    Done = 1;
373|    ++ Numticks;
374|
375|    for( i = 0, mp = Measure; ++i <= NUM_TRACKS ; mp++ )
376|    {
377|        if( (p = *mp)->num_beats > 0 )
378|        {
379|            if( p->cur_tick==p->ticks_per_beat )
380|            {
381|                /*  Ring bell on first tick of measure
382|                 *  unless this is a silent measure.
383|                 *  Warnings take precedence over
384|                 *  everything.
385|                 */
386|
387|                if( p->warning )
388|                {
389|                        Ring_bell = WARNING ;
390|                        p->warning = 0;
391|                }
392|
393|                else if( !p->silent )
394|                {
395|                    if( !Ring_bell )
396|                        Ring_bell = i;
397|                    else
398|                        Collision = i;
399|                }
400|            }
401|
402|            if( -- p->cur_tick <= 0 )
403|            {
404|                if( -- p->num_beats <= 0 )
405|                {
406|                    if( i == 1 )
407|                        ++Downbeat;
408|
409|                    ++( *mp );   /* go to next measure */
410|                }
411|                else
412|                {
413|                    p->cur_tick = p->ticks_per_beat;
414|
415|                    if( p->remainder > 0 )
416|                    {
```

(continued on next page)

Listing Seven *(Listing continued, text begins on page 106.)*

```
                        -- p->remainder ;
                        ++ p->cur_tick  ;
                    }
                }
            }
            Done = 0;
        }
    }
}

#pragma check_stack+

/*-------------------------------------------------------------*/

on_break()
{
        /* Routine for signal(), tries to put the clock rate
         * back to normal on a Ctrl-Break. Note that this
         * routine can fail if Ctrl-Break is hit several times
         * in quick succession.
         */

        signal( SIGINT, SIG_IGN );
        slowdown();
        exit(0);
}

/*-------------------------------------------------------------*/

print_stats()
{
}

/*-------------------------------------------------------------*/

usage()
{
        char    **p;
        for( p = Usage_msg; *p; fprintf(stderr,"%s\n", *p++) )
                ;
        exit(1);
}

/*-------------------------------------------------------------*/

main( argc, argv )
char    **argv;
{
        static int measure = 0; /* current measure in track 0 */
        static int dull    = 0; /* Dull output                */
        static int stats   = 0; /* Statistics only            */
        static int i;

        if( argc == 3 )
        {
            --argc;

            if( **(++argv) != '-' )
                usage();

            switch( argv[0][1] )
            {
            case 'd': dull  = 1;        break;
            default:                    usage();
            }
        }
        else if( argc != 2 || *argv[1] == '-' )
                usage();

        if( !build_tracks( argv[1] ) )
                exit( 2 );

        if( stats )
```

```
490|        {
491|                print_stats();
492|                exit(0);
493|        }
494|
495|        signal( SIGINT, on_break );
496|
497|        for( speedup(FACTOR, timer_intr); !Done; )
498|        {
499|            cli();              /* Interrupts off     */
500|
501|            if( Ring_bell == WARNING )
502|            {
503|                Collision  = 0;
504|                Ring_bell  = 0;
505|                sti();
506|                beep ( C4*8,  0.125 );
507|                delay(        0.1   );
508|                beep ( C4*8,  0.125 );
509|            }
510|            else if( Collision )
511|            {
512|                /* the higher track wins */
513|
514|                i = max( Collision, Ring_bell );
515|                Collision = 0;
516|                Ring_bell = 0;
517|                sti();
518|                beep( dull ? C4*2 : C4 * i, 0.125 );
519|            }
520|            else if( Ring_bell )
521|            {
522|                /* You must set Ring_bell to
523|                 * 0 before calling beep so that
524|                 * a note won't collide with itself.
525|                 */
```

(continued on next page)

Listing Seven *(Listing continued, text begins on page 106.)*

```
526|
527|                 i = Ring_bell;
528|                 Ring_bell = 0;
529|                 sti();
530|                 beep( dull ? C4*2 : C4 * i, 0.125 );
531|             }
532|             else
533|             {
534|                     sti();
535|             }
536|
537|
538|             if( Downbeat != measure )
539|                 printf( "\r%d", measure = Downbeat );
540|         }
541|
542|         printf( "\rDone" );
543|
544|         D( printf("%d ticks overall\n", Numticks ); )
545|         slowdown();
546|         printf("\n");
547| }
```

End Listings

The Software Family

649 Mission Street
San Francisco, CA 94105

(1)800-443-7176

In California or outside U.S.

(415)583-4166

Trust TSF to provide the best value and service!

- Technical advice and support by programmers
- Honest and equitable business practices
- Prompt, flexible service to meet your needs

We accept checks, Visa, MasterCard, American Express and COD. We charge your card only when we ship and do **not** add a surcharge. We provide free UPS delivery on software orders over $100 ($3 delivery charge on orders under $100). We add actual charges for UPS Blue Label or Federal Express rush service. Our COD fee is $2. We allow return privileges on most products. We carry a large inventory and ship promptly, so you receive your shipment within 3 to 6 working days of placing your order. Give us a try.

Turbo C Specials

Borland **Turbo C** (list $99) **$59**

Blaise **Turbo C Tools** (list $129) **$95**
ISR/TSR support, critical error trapping, direct video access, more

Creative **Vitamin C** (list $225) **$150**
Windows, menus and data entry with source

Gimpel **PC Lint** (list $139) **$95**
Checks all modules of a C program to warn if function calls do not include the proper number and types of parameters, if global variables are declared inconsistently or if you have used common C "gotchas"

Metagraphics **Turbo Window/C** ($95) **$75**
Graphics windows, menus and multiple fonts

Softfocus **Btree & ISAM** (list $115) **$99**
Complete multi-key ISAM file system with source

Prices good until October 15, 1987

Basic **C**
Turbo Pascal **dBase**
Publishing **AI**

Aldebaran **Source Print** (list $75)	$59
Blaise **C Tools +** (list$175)	$125
Blaise **Light Tools** For Datalight (list $100)	$79
Blaise **Turbo Power Tools +** (list $99)	$79
Borland **Turbo Basic** (list $99)	$61
Borland **Turbo C** (list $99)	$59
Bytel **Genifer** dbase III application generator ($395)	$225
Comtel **dbScope** Interactive dBase debugger ($70)	$62
Comtel **dbTools** (list $70)	$62
Creative **Vitamin C** Specify compiler (list $225)	$150
Creative Programming **VC Screen** (list $100)	$79
Datalight **Optimum C** w/Easy Edit (list $140)	$99
Fox **FoxBase +** fast dBase compiler ($395)	$249
Gimpel **C-Terp** Specify compiler (list $300)	$224
Gimpel **PC-Lint** (list $139)	$95
Greenleaf Data Windows Specify compiler ($225)	$157
Guidelines **C + +** for Microsoft C (list $195)	$175
Lattice **C** (list $500)	$270
Matrix **Synergy Development Toolkit** (list $395)	$309
MicroHelp **Mach2 for Microsoft Basic** ($69) windows, fast i/o, critical error trapping, more	$55
MicroHelp **Mach2 for Turbo Basic** ($69)	$55
MicroHelp **Mach2 for Turbo Pascal** ($69)	$55
Microsoft **C v5** w/Codeview & Quick C (list $450)	$279
Microsoft **Quick C** (list $99)	$68
Microsoft **Fortran** w/Codeview (list $450)	$270
Microsoft **Quick Basic v3** (list $99)	$65
MKS **Toolkit** Korn shell, vi, grep, 30 more ($139)	$109
Periscope **Periscope I** (list $345)	$279
Periscope **Periscope II** (list $175)	$139
Periscope **Periscope III** (list $995)	$799
Periscope **Periscope III 10Mhz** (list $1095)	$876
Phoenix **PFix-86 +** (list $395)	$250
Phoenix **Plink-86 +** (list $495)	$315
Tom Rettig **Library** (list $99)	$87
Turbo Power **Extender** (list $85)	$68
Turbo Power **Optimizer** (list $75)	$59
Turbo Power **Optimizer** w/Source (list $175)	$99
Turbo Power **TDebug Plus v2** (list $60)	$48
Turbo Power **Turbo Utilities** w/source ($95)	$76
Unipress **Phact** RDBMS w/ report & query ($249)	$199
Wallsoft **UI Programmer** (list $295)	$260
Wordtech **dBXL** dBase III work-alike (list $169)	$110

TSF carries hundreds of products from dozens of publishers. Call or write for a FREE catalog or a price quote.

Listing One *(Text begins on page 122.)*

```
PROGRAM  CBTree;

{=====================================================
  Program to test the compare the time needed in creating and
   searching a binary tree and a CB-tree.
  Author: Namir Clement Shammas
=====================================================}

CONST SIZE = 1000;
      RANGE = 10000;
      MainLoopCount = 100;

TYPE Bin_Ptr =  ^Bin_Node;

     { normal binary tree strcutures }
     Bin_Node = RECORD
              Value : INTEGER;
              Left, Right : Bin_Ptr;
            END;

     CB_Ptr =  ^CB_Node;

     { Record structure for clustured binary tree }
     CB_Node = RECORD
              Value : INTEGER;
              HLeft, HRight,
              LLeft, LRight : CB_Ptr;
            END;

     REGTYPE = RECORD
                 AX,BX,CX,DX,BP,
                 DI,SI,DS,ED,FLAGS  : INTEGER
               END;

     Time_Rec = RECORD
                  HOUR, MIN, SEC, HSEC : BYTE
                END;


VAR Numbers : ARRAY [1..SIZE] OF INTEGER;
    Iter, I, Choice, CDV, Diff : INTEGER;
    BinTreeRoot : Bin_Ptr;
    CBTreeRoot : CB_Ptr;
    Timer_Start, Timer_Stop, Time_Elapsed : Time_Rec;

{-------------------------------------------------------}

PROCEDURE Get_Time(VAR TIME : Time_Rec { output });

VAR REGISTER : REGTYPE;
    AH       : BYTE;

BEGIN

  AH  :=  $2C;

     WITH REGISTER, TIME DO BEGIN
        AX:= AH SHL 8;
        MSDOS(REGISTER);
        HOUR :=  Hi(CX);
        MIN  :=  Lo(CX);
        SEC  :=  Hi(DX);
        HSEC :=  Lo(DX);
      END;
END;

{-------------------------------------------------------}

PROCEDURE Time_Diff(T_Start,
                    T_Finish       : Time_Rec;  { input  }
                    VAR Delta_Time : Time_Rec   { output });
```

```
BEGIN

 WITH Delta_Time DO BEGIN

    HOUR := T_Finish.HOUR - T_Start.HOUR;

    IF T_Start.MIN > T_Finish.MIN THEN BEGIN
        HOUR := HOUR - 1;
        T_Finish.MIN := T_Finish.MIN + 60;
    END;
    MIN := T_Finish.MIN - T_Start.MIN;

    IF T_Start.SEC > T_Finish.SEC THEN BEGIN
        MIN := MIN - 1;
        T_Finish.SEC := T_Finish.SEC + 60;
    END;
    SEC := T_Finish.SEC - T_Start.SEC;

    IF T_Start.HSEC > T_Finish.HSEC THEN BEGIN
        SEC := SEC - 1;
        T_Finish.HSEC := T_Finish.HSEC + 100;
    END;
    HSEC := T_Finish.HSEC - T_Start.HSEC;

 END; (* WITH *)

END; (* Time_Diff *)

{-------------------------------------------------------}

PROCEDURE Show_Time(T : Time_Rec { input });

BEGIN
    WITH T DO BEGIN
        WRITE('Time elapsed = ',HOUR,' : ',MIN,' : ',SEC,'.',HSEC);
    END;
END; (* Show_Time *)

{-------------------------------------------------------}

PROCEDURE Create(Choice : INTEGER { input });

VAR J : INTEGER;
    Angle, FloatSize, Increment : REAL;

BEGIN
    CASE Choice OF
     1 : BEGIN
            Angle := 0.0;
            Increment := Pi / 360.0;
            FOR J := 1 TO SIZE DO BEGIN
                Numbers[J] := Trunc(SIN(Angle) * RANGE);
                Angle := Angle + Increment;
            END;
        END;
     2 : BEGIN
            Angle := 0.0;
            Increment := Pi / 360.0;
            FOR J := 1 TO SIZE DO BEGIN
                Numbers[J] := Trunc(COS(Angle) * RANGE);
                Angle := Angle + Increment;
            END;
        END;

     ELSE BEGIN

            Numbers[1] := RANGE div 2;
            FOR J := 2 TO SIZE DO
                Numbers[J] := Trunc(Random * RANGE);
        END;
    END; { CASE }
 END;
```

(continued on next page)

Listing One *(Listing continued, text begins on page 122.)*

```
{-------------------------------------------------------}

PROCEDURE Bin_Insert(VAR Root : Bin_Ptr; { input }
                         Item : INTEGER  { input });
(* Insert element in binary-tree *)
BEGIN
    IF Root = NIL THEN BEGIN
        NEW(Root);
        Root^.Value := Item;
        Root^.Left := NIL;
        Root^.Right := NIL
    END
    ELSE
        WITH Root^ DO
            IF Item < Value THEN Bin_Insert(Left,Item)
                            ELSE Bin_Insert(Right,Item);
END;

{-------------------------------------------------------}

PROCEDURE Bin_Search(VAR Root : Bin_Ptr; { input }
                        Target : INTEGER  { input });
(* Recursive procedure to search for Target value *)
BEGIN
    IF Root <> NIL THEN
        IF Target <> Root^.Value THEN
            IF Target < Root^.Value THEN BEGIN
                Root := Root^.Left;
                Bin_Search(Root,Target)
            END
            ELSE BEGIN
                Root := Root^.Right;
                Bin_Search(Root,Target)
            END;
END;

{-------------------------------------------------------}

PROCEDURE CB_Insert(VAR Root : CB_Ptr; { input }
                    VAR Item : INTEGER { input });
(* Insert element in a CB-tree *)


BEGIN
    IF Root = NIL THEN BEGIN
        NEW(Root);
        Root^.Value  := Item;
        Root^.LLeft  := NIL;
        Root^.LRight := NIL;
        Root^.HLeft  := NIL;
        Root^.HRight := NIL;
    END
    ELSE
        WITH Root^ DO BEGIN
            Diff := Value - Item;
            IF Diff > 0 THEN
                IF ABS(Diff) <= CDV THEN CB_Insert(LLeft,Item)
                                    ELSE CB_Insert(HLeft,Item)
            ELSE
                IF ABS(Diff) <= CDV THEN CB_Insert(LRight,Item)
                                    ELSE CB_Insert(HRight,Item);
        END; (* WITH *)
END;

{-------------------------------------------------------}

PROCEDURE CB_Search(VAR Root   : CB_Ptr; { input }
                    VAR Target : INTEGER { input });
(* Recursive procedure to search for Target value *)


BEGIN
```

```
    IF Root <> NIL THEN
        IF Target <> Root^.Value THEN BEGIN
            Diff := Root^.Value - Target;
            IF Target < Root^.Value THEN BEGIN
                IF ABS(Diff) <= CDV THEN Root := Root^.LLeft
                                    ELSE Root := Root^.HLeft;
                CB_Search(Root,Target)
            END
            ELSE BEGIN
                IF ABS(Diff) <= CDV THEN Root := Root^.LRight
                                    ELSE Root := Root^.HRight;
                CB_Search(Root,Target)
            END;
        END;
END;

{-------------------------------------------------------}

BEGIN (* MAIN *)
  CDV := Trunc(0.05 * RANGE);

  ClrScr;
  WRITE(' ':10);
  WRITELN('--------- Menu for Method of Generating Numbers --------');
  WRITELN;
  WRITELN(' 0) Random numbers ');
  WRITELN(' 1) Sine function ');
  WRITELN(' 2) Cosine function ');
  WRITELN;
  WRITE('Enter code for array creation : ');
  READLN(Choice); WRITELN;
  Create(Choice);

  WRITELN; WRITELN('Created array'); WRITELN;
  (* Building the binary tree *)
  BinTreeRoot := NIL;
  Get_Time(Timer_Start);
  FOR I := 1 TO SIZE DO
     Bin_Insert(BinTreeRoot,Numbers[I]);
  Get_Time(Timer_Stop);
  Time_Diff(Timer_Start, Timer_Stop, Time_Elapsed);
  Show_Time(Time_Elapsed);
  WRITELN(' for creating binary Tree'); WRITELN;

  (* Building the CB-tree *)
  CBTreeRoot := NIL;
  Get_Time(Timer_Start);
  FOR I := 1 TO SIZE DO
     CB_Insert(CBTreeRoot,Numbers[I]);
  Get_Time(Timer_Stop);
  Time_Diff(Timer_Start, Timer_Stop, Time_Elapsed);
  Show_Time(Time_Elapsed);
  WRITELN(' for creating CB-Tree'); WRITELN;

  Get_Time(Timer_Start);
  FOR Iter := 1 TO MainLoopCount DO
      FOR I := SIZE DOWNTO 1 DO
          Bin_Search(BinTreeRoot,Numbers[SIZE + 1 - I]);
  Get_Time(Timer_Stop);
  Time_Diff(Timer_Start, Timer_Stop, Time_Elapsed);
  Show_Time(Time_Elapsed); WRITELN(' for binary tree search');
  WRITELN;

  Get_Time(Timer_Start);
  FOR Iter := 1 TO MainLoopCount DO
         FOR I := SIZE DOWNTO 1 DO
          CB_Search(CBTreeRoot,Numbers[SIZE + 1 - I]);
  Get_Time(Timer_Stop);
  Time_Diff(Timer_Start, Timer_Stop, Time_Elapsed);
  Show_Time(Time_Elapsed); WRITELN(' for CB-tree search');
  WRITELN;
  WRITELN('DONE'); WRITELN;
END.
```

End Listing One

(Listing Two begins on next page.)

Listing Two *(Text begins on page 122.)*

```
PROGRAM Test_Clustered_Lists;

{$R+,V-}

{============================================================

  Turbo Pascal program that implements routine for inserting,
  searching and viewing clustered list structures.

  Copyright (c) 1987  Namir Clement Shammas

============================================================}

CONST LENSTRING = 30;
      MAX_LIST = 100;
 MDNMØ
      CDV = 0; { Critical Difference value }

TYPE LSTRING = STRING[LENSTRING];
     KeyArray = ARRAY [1..MAX_LIST] OF LSTRING;

     ListPtr = ^ListNode;

     { CLustered List structure }
     ListNode = RECORD
                  Key : LSTRING;
                  { other fields may be placed here }
                  NextPtr, NextHi : ListPtr;
                END;

VAR Head : ListPtr;
    LesKeys : KeyArray;
    I, Count : INTEGER;

{-------------------------------------------------------}

PROCEDURE Search_Node(Head        : ListPtr; { input }
                      SearchData  : LSTRING; { input }
                  VAR Found       : BOOLEAN; { output }
                  VAR LastPtr,
                      ThisPtr     : ListPtr  { output });

{ search for 'SearchData' in list }

VAR HiTrack : BOOLEAN;
    Ord1, Diff : INTEGER;

BEGIN
    Found := FALSE;
    HiTrack := TRUE;
    LastPtr := NIL;
    ThisPtr := Head;

    Ord1 := ORD(SearchData[1]);

    WHILE (ThisPtr <> NIL) AND (ThisPtr^.Key < SearchData) DO BEGIN

        LastPtr := ThisPtr;

        IF HiTrack THEN BEGIN

            Diff := ORD(ThisPtr^.Key[1]) - Ord1;

            IF ABS(Diff) > CDV
            THEN
                ThisPtr := ThisPtr^.NextHi

            ELSE BEGIN
                ThisPtr := ThisPtr^.NextPtr;
                HiTrack := FALSE { switch to low track }
```

```
            END; { IF ABS(Diff) }
        END

        ELSE
            ThisPtr := ThisPtr^.NextPtr;
        { END IF HiTrack }

    END; { WHILE }

    IF ThisPtr <> NIL THEN Found := (ThisPtr^.Key = SearchData);

END; { Search_Node }

{-------------------------------------------------------}

PROCEDURE Insert_List(VAR Head : ListPtr; { in/out }
                      NewData : LSTRING  { input  });
{ Insert new data string into the list }

VAR Found : BOOLEAN;
    Ord1, Diff : INTEGER;
    Tempo : LSTRING;
    Node, LastPtr, ThisPtr : ListPtr;

BEGIN

    Ord1 := ORD(NewData[1]); { get ascii code of firt char }

    IF Head = NIL THEN BEGIN { start a new list }
        new(Head);
        WITH Head^ DO BEGIN
            NextPtr := NIL;
            NextHi  := NIL;
            Key := NewData
        END; { WITH }
    END

    ELSE BEGIN { expand list }
        new(Node);
        WITH Node^ DO BEGIN
            Key := NewData;
            NextPtr := NIL;
            NextHi := NIL
        END; { WITH }

        Search_Node(Head, Node^.Key, Found, LastPtr, ThisPtr);

        IF LastPtr = NIL THEN BEGIN { insert as new list head }
            Diff := ORD(Head^.Key[1]) - Ord1;
            IF ABS(DIFF) > CDV THEN
                Node^.NextHi := Head
            ELSE
                Node^.NextHi  := Head^.NextHi;
                Node^.NextPtr := Head;
            { END IF }

            Head := Node;
        END

        ELSE BEGIN { insert new data in the middle or at the tail }

            Diff := Ord1 - ORD(LastPtr^.Key[1]);

            IF Diff <= CDV THEN BEGIN
                { insert inside a clustered sublist }
                { LasPtr may be a high of low track node }
                Node^.NextPtr := LastPtr^.NextPtr;
                LastPtr^.NextPtr := Node
            END
            ELSE BEGIN
```

(continued on next page)

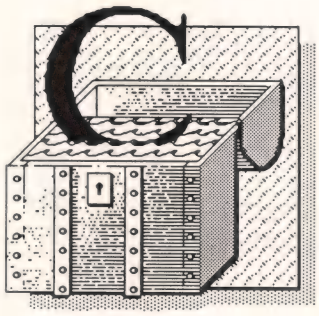
Listing Two *(Listing continued, text begins on page 122.)*

```
                IF ThisPtr <> NIL
                THEN BEGIN
                    Diff := Ord1 - ORD(ThisPtr^.Key[1]);
                    IF ABS(Diff) > CDV
                    THEN BEGIN {insert between two high track nodes }
                        Node^.NextHi := LastPtr^.NextHi;
                        LastPtr^.NextHi := Node
                    END
                    ELSE BEGIN {swap names in the next high track node }
                        Tempo := Node^.Key;
                        Node^.Key := ThisPtr^.Key;
                        ThisPtr^.Key := Tempo;
                        { insert a new swapped first element }
                        { in clustered sublist               }
                        Node^.NextPtr := ThisPtr^.NextPtr;
                        ThisPtr^.NextPtr := Node
                    END; { IF }

                END
                ELSE BEGIN { insert as last high track node }
                    Node^.NextHi := LastPtr^.NextHi;
                    LastPtr^.NextHi := Node
                END; { IF }
            END; { IF }
        END; { IF LastPtr = NIL }

    END; { IF Head = NIL }

END; { Insert_List }

{-------------------------------------------------------}

PROCEDURE List_to_Array(Head      : ListPtr;  { input  }
                        VAR Keys  : KeyArray; { output }
                        VAR Count : INTEGER   { output });

{ Converts the list to an array containing sorted names }

{-------------------------------------------------------}

PROCEDURE Visit_Low_Node(VAR Node : ListPtr);
{ Local recursive routine to visit low tracks of a list }

BEGIN

    IF (Node <> NIL) AND (Count < MAX_LIST) THEN BEGIN
        Count := Count + 1;
        Keys[Count] := Node^.Key;
        WRITE(' ',Keys[Count]:10);
        Visit_Low_Node(Node^.NextPtr);
    END
    ELSE WRITELN(' -]');

END; { Visit_Low_Node }

{-------------------------------------------------------}

PROCEDURE Visit_Hi_Node(VAR Node : ListPtr);
{ Local recursive routine to visit high tracks of a list }

BEGIN

    IF (Node <> NIL) AND (Count < MAX_LIST) THEN BEGIN
        Count := Count + 1;
        Keys[Count] := Node^.Key;
        WRITE(Keys[Count]:10);
        Visit_Low_Node(Node^.NextPtr);
        Visit_Hi_Node(Node^.NextHi);
    END;
```

```
END; { Visit_Hi_Node }

{-------------------------------------------------------}

BEGIN


    IF Head <> NIL THEN BEGIN
        Count := 0;
        Visit_Hi_Node(Head);

    END

    ELSE
        Count := 0;
    { END IF }


END; { List_to_Array }

{-------------------------------------------------------}

BEGIN

    ClrScr;
    IF CDV < 0 THEN BEGIN
        WRITELN('Adjust Critical Difference Value Please');
        HALT
    END;

    Head := NIL;
    WRITELN('List of sorted capitals '); WRITELN;
    Insert_List(Head,'Athens');      Insert_List(Head,'London');
    Insert_List(Head,'Bonn');        Insert_List(Head,'Ankara');
    Insert_List(Head,'Sau Paulo');   Insert_List(Head,'Moscow');
    Insert_List(Head,'Otawa');       Insert_List(Head,'Tokyo');
    Insert_List(Head,'Bern');        Insert_List(Head,'Warsaw');
    Insert_List(Head,'Cairo');       Insert_List(Head,'Rome');
    Insert_List(Head,'Madrid');      Insert_List(Head,'Lisbon');
    Insert_List(Head,'Paris');       Insert_List(Head,'Baghdad');

    List_to_Array(Head, LesKeys, Count);

END.
```

End Listings

The Ultimate Metronome: Writing Interrupt Service Routines in C



The ostensible topic of this month's column is a fancy metronome program capable of doing polyrhythms and even more complex rhythmic patterns. As is often the case, however, the pieces are as interesting as the whole. For example, in order to get adequate timing resolution, I had to speed up the IBM PC's system clock and then supply my own interrupt service routine, written largely in C—though there's a little assembly language. The routines used for this purpose are useful in other applications as well (such as a preemptive multitasking scheduler that I'm also writing).

The metronome is pretty spiffy in its own right, though. I use it for my own compositional work to create "click tracks" for electronic pieces. (A click track is one track of a multitrack tape that holds nothing but timing information. You use it as a reference as you add additional tracks to the piece and then remove it from the final mix.) The program lets you plot all the complex rhythmic patterns required for a complete piece of music. (The maximum time varies; the program can do up to 1,280 measures, and if these are measures of 6/8 at metronome 120, that's 64 minutes.) Though I'm just ringing the bell on the PC to mark beats, it wouldn't be too hard to modify this program to use a MIDI drum machine or the like to make the sounds.

by Allen Holub

Metronomes and Time Signatures

When metronomes were introduced in the early 19th century (Beethoven was the first major composer to use one), it finally became possible for composers to present timing information accurately to performers. To understand why this wasn't possible before, you have to look at how a standard time signature works. Music, by nature, is cyclical, and a measure is one complete cycle of the underlying rhythmic pattern. A march, for example, has two beats in a measure (oom pah, oom pah), a waltz has three (oom pah pah, oom paa pah), and rock and roll usually has four-beat measures. The main purpose of the measure structure is to tell the performer where to put emphasis. That is, the first note of the measure (called the downbeat) is usually played a little more forcefully than the other notes in the measure (OOM! pah pah).

Coupled with the measure are the individual notes. A whole note requires an entire measure to play. If you think in terms of an organ, you hold the key down for the entire measure. A half note requires half a measure. There are two half notes in a whole note, two quarter notes in a half note, two eighth notes in a quarter note, and so forth. Each of these types of notes are represented by a slightly different symbol in a musical score. You'll note (so to speak) that this system is self-relative. There's no absolute durational information in a particular note symbol—everything's relative to the other notes.

A standard time signature gives you two pieces of information—the number of beats in a measure and which of the various types of notes in the score represent one beat. For example, the time signature 3/4 (pronounced "three four") has three beats in the measure and a quarter note is one beat long. So a complete three-beat measure can be composed of three quarter notes, six eighth notes, two quarter notes and two eighth notes, and so forth. A time signature is not a fraction; 6/8 is not the same thing as 3/4. The emphasis in the former is on every sixth beat, whereas the emphasis in the latter is on every third beat. There's no information in a time signature about the actual duration of a beat in seconds. This duration is usually detailed with written instructions—most often in Italian—that are quite inexact (fast, slow, medium-fast, and so on).

The metronome changed all this. The mechanical metronome used by Beethoven is still in use today. It consists of a spring-driven pendulum with a weight on it. The position of the weight determines the rate at

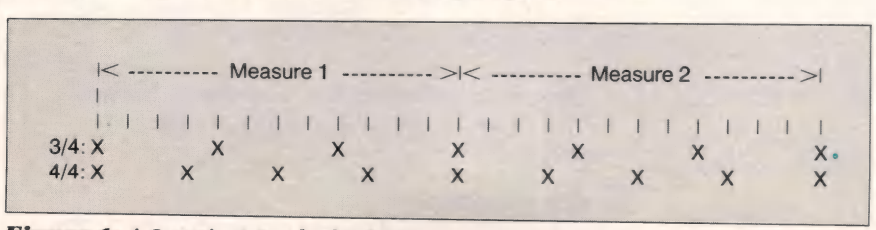


Figure 1: *A 3 against 4 polyrhythm*

Announcing - the database development system that you designed.

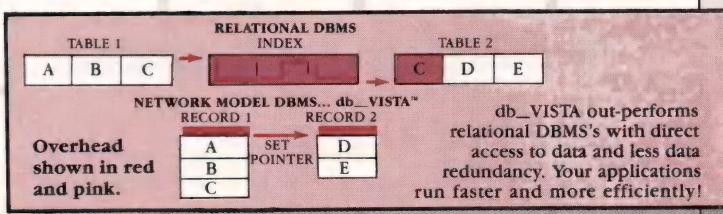
db_VISTA III™

C PROGRAMMERS- We asked what you wanted in a database development system and we built it!

db_VISTA III™ is the database development system for programmers who want powerful, high performance DBMS capabilities ... and in any environment. Based on the network database model and the B-tree indexing method, db_VISTA III gives you the most powerful and efficient system for data organization and access. From simple file management to complex database structures with millions of records. db_VISTA III runs on most computers and operating systems like MS-DOS, UNIX, VAX/VMS and OS/2. It's written in C and the complete source code is available, so your application performance and portability are guaranteed! With db_VISTA III you can build applications for single-user microcomputers to multi-user LANs, up to minis and even mainframes.

RAIMA'S COMMITMENT TO YOU: No Royalties, Source Code Availability, 60 days FREE Technical Support and our 30-day Money-Back Guarantee. Extended services available include: Application Development, Product Development, Professional Consulting, Training Classes and Extended Application Development Support.

HOW TO ORDER: Purchase only those components you need. Start out with Single-user for MS-DOS then add components, upgrade ... or purchase Multi-user with Source for the entire db_VISTA III System. It's easy... call toll-free today!



The db_VISTA III™ Database Development System

1 db_VISTA™: The High Performance DBMS

The major features include:

- Multi-user support for LANs and multi-user computers.
- Multiple database access.
- File and record locking.
- Automatic database recovery.
- Transaction processing and logging.
- Timestamping.
- Database consistency check utility.
- Fast access methods based on the network database model and B-tree indexing. Uses both direct "set" relations and B-tree indexing independently for design flexibility and performance.
- An easy-to-use interactive database access utility.
- File transfer utilities for importing/exporting ASCII text and dBASE II/III files.
- A Database Definition Language patterned after C.
- Virtual memory disk caching for fast database access.
- A runtime library of over 100 functions.
- **Operating systems:** MS-DOS, UNIX V, XENIX, VMS, OS/2.
- **C Compilers:** Lattice, Microsoft, IBM, Aztec, Computer Innovations, Turbo C, XENIX, and UNIX.
- **LAN systems:** LifeNet, NetWare, PC Network, 3Com, SCO XENIX-NET, other NETBIOS compatible MS-DOS networks.

2 db_QUERY:™ The SQL-based Query.

- Provides relational view of db_VISTA applications.
- Structured Query Language
- C linkable.
- Predefine query procedures or run ad-hoc queries "on the fly".

3 db_REVISE™: The Database Restructure Program.

- Redesign your database easily.
- Converts all existing data to revised design.

All components feature royalty-free run-time distribution, source code availability and our commitment to customer service. That's why corporations like ARCO, AT&T, Hewlett-Packard, IBM, Northwestern Mutual Life, UNISYS and others use our products.

db_VISTA III™ Database Development System

db_VISTA III™	$595 - 3960
db_QUERY™	$595 - 3960
db_REVISE™	$595 - 3960
db_VISTA™ File Manager	Starts at $195

We'll answer your questions, help determine your needs and get you started.

CALL TODAY!
1-800-db-RAIMA
(that's 1-800-327-2462)

MasterCard VISA

RAIMA™ CORPORATION

3055 112th Avenue N.E., Bellevue, WA 98004 **(206)828-4636**
Telex: 6503018237MCIUW FAX: (206)828-3131

DD97

(continued from page 106)

which the pendulum swings, and the box ticks at the end of each swing. Metronomes are calibrated in beats per minute, so a metronome 60 is one beat per second and a metronome 120 is two beats per second. The composer can now put an instruction at the top of the score saying something such as "an eighth note is played at metronome 100."

Traditional metronomes are fine if you're writing waltzes, but they aren't really adequate for most modern music for three reasons. First, some pieces have different time signatures on literally every measure, and you obviously can't stop after every measure to adjust your metronome. A second, related, problem is a piece with measures of alternating time signatures—say, alternating measures of 5/8 and 6/8. Dave Brubeck and Steve Reich both use this technique extensively. Finally, there's the problem of polyrhythms. A polyrhythm is a rhythmic pattern in which two dissimilar time signatures are superimposed. For example, in three against four, a three-beat pattern is played in one hand and a four-beat pattern in the other. These come together in at least one place (usually on the downbeat) of every measure. This timing is illustrated in Figure 1, page 106. The three-beat pattern beats on every fourth tick, the four-beat pattern on every third tick. Polyrhythmic music is usually great dance music—you hear it in Latin, Brazilian, and West African pop music and more and more in contemporary Western music.

Click: A Four-Channel Programmable Metronome

The program I present this month, called click, is a programmable metronome that solves all the problems I've just discussed. Its invocation syntax is:

```
click [-d] file
```

where *file* holds an input program. Click outputs two things—beats in the form of sounds and a running count of the number of measures played so far.

Click is programmed using a multi-

track tape recorder model. Four tracks (numbered 0 to 3) are supported, each of which can contain up to 1,280 measures. Track 0 is always required; the others are optional. The four tracks can be programmed independently, or they can synchronize to each other at the beginning of each measure. Different notes are used for the different tracks so that you can distinguish them. Track 0 uses middle C, track 1 uses the C one octave up, track 2 uses the G above that, and track 3 uses the C above that. When there's a conflict (a beat occurring at the same time on more than one track), the note associated with the higher track is used. The optional *-d* command-line switch forces all four tracks to use the same note (C5) for their beats.

A short click program is shown in Example 1, page 112. The input language works as follows. White space (spaces, tabs, blank lines, and so on) is ignored except as is needed to separate commands. By the same token, all lines that don't start with the word *track* and all text on a line that follows a semicolon are ignored. You can use this mechanism to comment your programs. The virtual tape is assembled using *track* commands, which take the following form:

track n: <measure> [, <measure>]
...

where *n* is a number in the range 0–3 that specifies a track on the virtual tape. A *<measure>* is all text starting with a colon or comma, up to the next comma or end of line. Several comma-delimited measures can be placed on a single line. Tracks are accumulated by successively adding the information in each *track* command to the indicated track.

A measure comprises one or more of the following commands, where *n* and *m* are numbers, and the commands can occur in any order:

n/m—A command that starts with a digit represents either the time signature or the number of beats in the measure. Here, *n* is the number of beats in the current measure and */m* is ignored. In fact, you can omit the */m* entirely if you like. A time signature is required in every measure—I'll show you some examples in a mo-

Avoid extra steps.

You've got better things to do than repeat the same steps. Over . . . and over . . . and over. Up your productivity with Greenleaf Software.

With more than 70 new functions added to our popular libraries, Greenleaf is now the most complete and mature C language function resource available. It's no wonder we've been rated the best. Winning program developers in major corporations such as IBM, EDS and GM have proven our reliability in thousands of applications.

Step Lively

New Greenleaf Functions v.3.10 includes 295 of the functions you've been asking for — DOS, disk, video, color text and graphics, string, time/date, keyboard, plus many more! With Greenleaf, you'll finish faster.

Cut Corners

When it comes to merging information, the new Greenleaf Comm Library v.2.10 is the fastest communications facility of its kind. Over 120 functions — ring buffered, interrupt-driven asynchronous communications. And, only Greenleaf gives you the power to build a 16-port communication system.

Get on the Fast Track

Order your new Greenleaf library today! See your dealer or call 1-800-523-9830.

Greenleaf Comm Library	$185.00
Greenleaf Functions	$185.00
Greenleaf DataWindows	$225.00
Greenleaf C Sampler	$ 94.50
Digiboard Comm4	$325.00
Digiboard Comm8	$535.00

In stock, shipped next day.

Greenleaf DataWindows and Turbo C

DataWindows, the finest C programming windows tool available, puts windows, transaction data entry and menus at your fingertips.

Our new TURBO C versions are ready to get you going fast! And, our new 3-in-1 C Sampler for only $94.50 supports both Turbo C and Quick C with comm, windows, menus and more! Our libraries support all popular C compilers for MS DOS.

GREENLEAF Software

Call Toll Free:

800-523-9830

In Texas and Alaska:

214-446-8641

Greenleaf Software, Inc.
16475 Dallas Parkway, Suite 570
Dallas, Texas 75248

CIRCLE 97 ON READER SERVICE CARD

(continued from page 109)

ment. The maximum number of beats in a measure (*n*) is 255.

@n—A number following an at sign (@) is used to set the metronome count. For example, you'd use the following to play 100 beats at metronome 120:

track 0: 100 @120

One measure of 3/4, with the quarter note at metronome 120, is represented as follows:

track 0: 3/4 @120

Alternating measures of 5/8 and 6/8 at different metronome counts can be done with the following:

track 0: 6/8 @120, 5/8 @100
track 0: 6/8 @120, 5/8 @100
track 0: 6/8 @120, 5/8 @100

Click supports a resolution as low as one metronome count—a metronome 1 is one beat/minute. There is no effective maximum count because the maximum is in the audio-frequency range.

The @ command is required for every measure on track 0. It's optional, however, on the other tracks. If the @ command is omitted, the time between beats is stretched out so that the measure takes up the same time as the corresponding measure on track 0. For example, you can take advantage of the higher note being used in a conflict to accent the downbeat of a measure as follows:

track 0: 3/4 @120, 3/4 @120
track 1: 1/4 , 1/4

Here, each measure of track 1 takes the same amount of time to play as the corresponding measure of track 0. You could do the same thing with:

track 0: 3/4 @120, 3/4 @120
track 1: 1/4 @ 30, 1/4 @ 30

but then you'd have to do the math yourself. This synchronization feature can also be used to do polyrhythms. A three against four polyrhythm can be specified with:

track 0: 3/4 @120, 3/4 @120
track 1: 4/4 , 4/4

Four beats on track 1 are played in the same amount of time that it takes to play three beats on track 0 (1½ seconds).

#n—A # sign followed by a number is a repeat count. For example, you could specify ten measures of 6/8 at metronome 100 with:

track 0: #10 6/8 @100

Note that 10 of the 1,280 measures are being used here. This isn't usually a problem, but if it is, the following command outputs the same number of beats but uses only one measure:

track 0: 60 @100

The earlier, two-measure-long, polyrhythm can be restated as:

track 0: #2 3/4 @120
track 1: #2 4/4

()—A measure surrounded by parentheses is counted silently—no sounds are output during that measure. This command is useful if you need measure-long rests in your music. For example, three measures of 3/4, followed by three measures of a three against four polyrhythm, followed by three more measures of 4/4, can be specified as follows:

track 0: #3 3/4 @100, #3 3/4 @100,

```
; Play four measures of 3/4, followed by four measures of a 3/4
; against 4/4 polyrhythm, followed by four measures of 3 against
; 4 against 5, followed by four measures of 3 against 4 against 5
; against 7. Sound a warning tone one measure before each change.

track 0: #4 3/4 @120 w, #4 3/4 @120 w, #4 3/4 @120 w, #4 3/4 @120 w
track 1: (#4 3/4),       #4 4/4,       #4 4/4,       #4 4/4
track 2: (#4 3/4),       (#4 5/4),     #4 5/4,       #4 5/4
track 3: (#4 3/4),       (#4 7/4),     (#4 7/4),     #4 7/4

; Now go into Steve Reich mode. Play 20 measures of 4/4 at
; metronome 150 and at the same time play 20 measure of 4/4
; at metronome 151. The clicks start out on the same
; beat and they very gradually go out of phase and then
; come back into phase again.

track 0: #20 4/4 @ 150
track 1: #20 4/4 @ 151
```

Example 1: *A click program*

(#3 3/4 @100)
track 1: (#3 4/4), #3 4/4 , #3 4/4

You could do the same thing with:

track 0: #3 3/4 @100, #3 3/4 @100
track 1: (#3 4/4), #6 4/4

warning—The *warning* command causes a warning tone (consisting of two high-pitched beeps) to be played rather than the normal downbeat of a measure.

If the measure is created with a *#n* command, only the last measure of the series is affected. For example, if you want to do four measures of 3/4, followed by a change to 6/8 but with a warning just before the change, you'd say:

track 0: #4 3/4 @120 warning, #10 6/8 @100

The program plays three measures of 3/4 as usual; then it plays a final measure of 3/4, but in this measure the warning tone is used for the

Flotsam and Jetsam

Multiple-Statement Macros

This month's Flotsam and Jetsam discusses how to write macros that have more than one statement in them. I've touched on some of this stuff in previous columns, but it's worthwhile to get everything in one place.

Two-statement macros such as:

```
#define X( ) stmt1( ); stmt2( )
```

are not usually a good idea. This macro, when used in:

```
if( condition )
   X( );
else
   something( );
```

expands as:

```
if( condition )
   stmt1( );
stmt2( );
else
   something( );
```

Adding curly braces does not solve the problem:

```
#define X( ) {stmt1( ); stmt2( );}
```

when used in:

```
if( condition )
   X( );
else
   something( );
```

expands to:

```
if( condition )
{
   stmt1( );
   stmt2( );
}
;
else
   something( );
```

The semicolon is a legitimate statement in C, so the *else* tries to bind to the preceding semicolon rather than the *if*.

One workable solution to the problem was used in the program I discussed this month, in which I've used an *if* statement to create a block. The general form is:

```
#define X( ) \
  if(1) \
  { \
     stmt1( ); \
     stmt2( ); \
  }else
```

You can even declare variables after the open curly brace if you like (see Listing Two, lines 50–61, page 82, for an example). These variables shadow (take precedence over) any variables that might have the same name in an outer block (they'll be different variables). The trailing, but empty, *else* clause is required. Without it, the following wouldn't work:

```
if( condition )
   X( );
else
   something( );
```

It would expand as:

```
if( condition )
   if(1)
   {
      ...
   }
   else
      something( );
```

The extra *else* statement forces the *else something( )* to bind properly.

A disadvantage of this method is that an *if* statement is not an expression. Consequently, you can't say *y = X( );* because it would evaluate to the illegal *y = if(1)...*. A way around this problem is to use the comma (or sequence) operator. The comma operator evaluates left to right and evaluates to the rightmost thing in the list. So:

```
#define X( ) \
  ( \
     stmt1( ), \
     stmt2( ) \
  )
```

executes *stmt1( )* first, then *stmt2( )*. The entire expression evaluates to the return value of *stmt2( )*. Note that I'm using parentheses, not braces, for grouping.

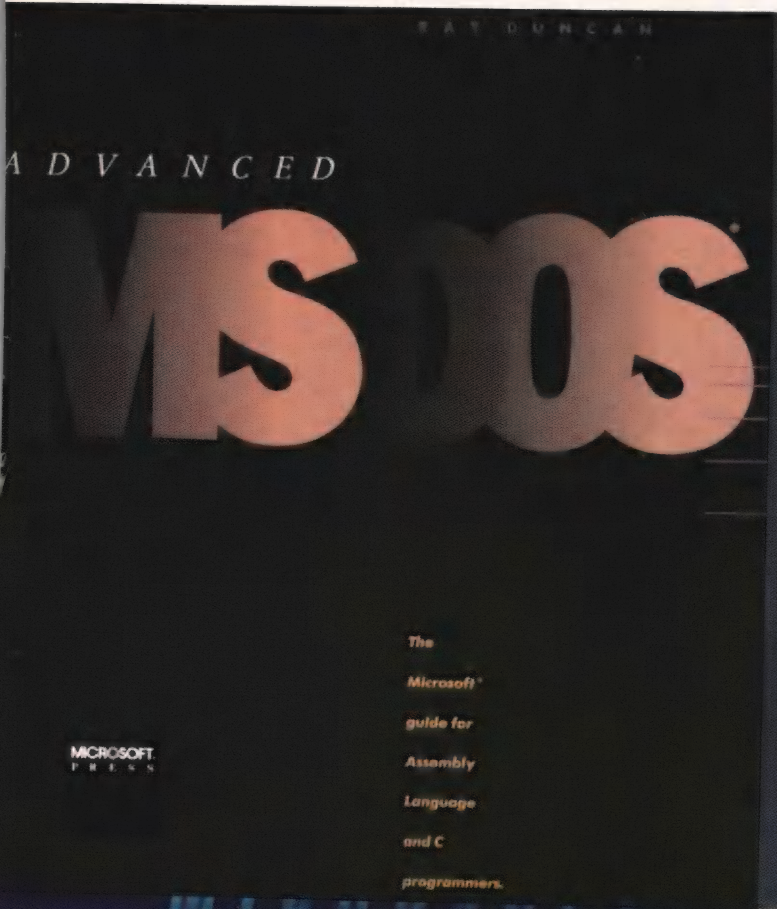
A disadvantage of the comma operator is that you can't declare variables local to the macro, as you can with braces.

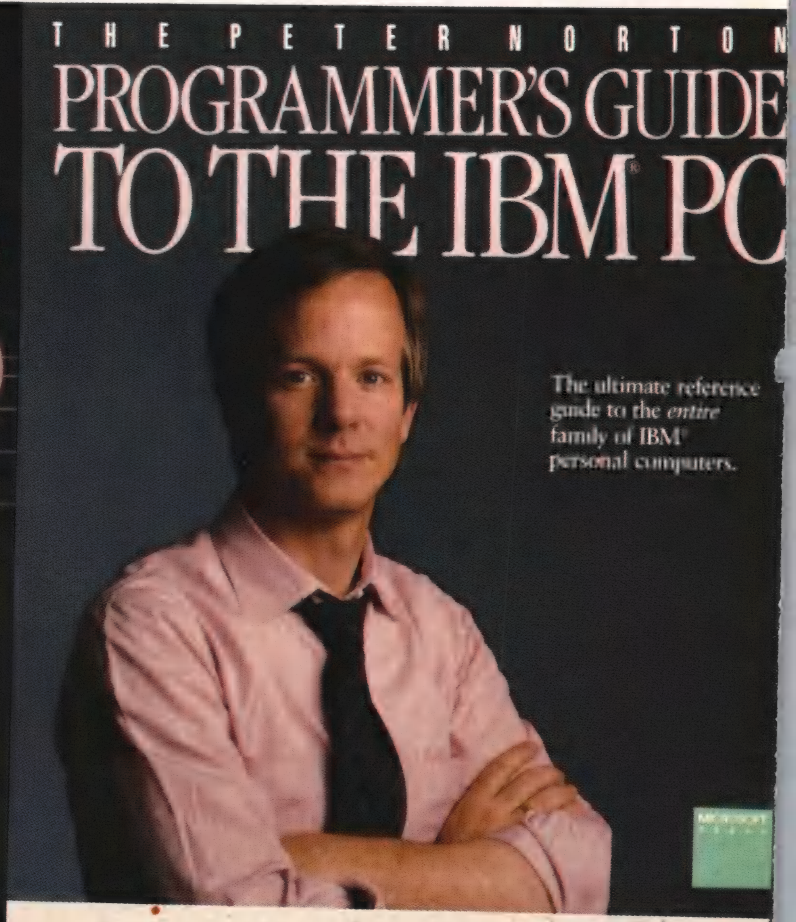
Don't confuse the comma operator with the comma that's used in a subroutine call. They're different things. In order to get a comma operator into a subroutine call, you have to surround the expression with parentheses, as in *foo((a,b), c)*. Here, the left comma is a comma operator, and the one on the right is an argument-list separator. This same caveat applies to the *D( )* macro that I discussed in the November 1986 Flotsam and Jetsam. The preprocessor accepts *D(printf("%d",x);)* but only because the comma is surrounded by parentheses. The following won't work:

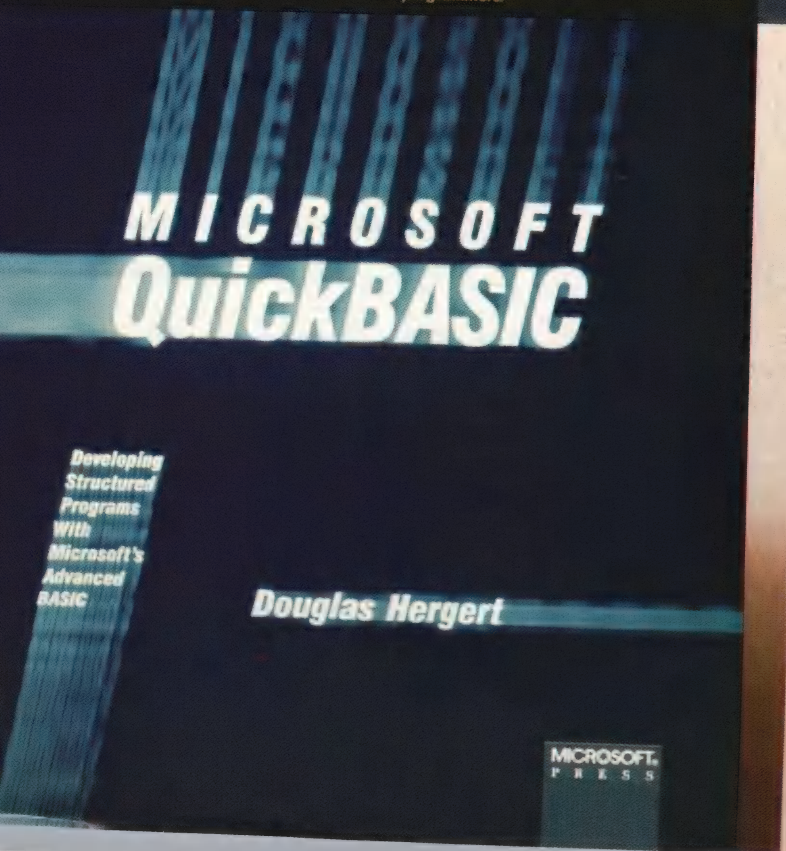
```
D( printf("%d", )
D( x); )
```

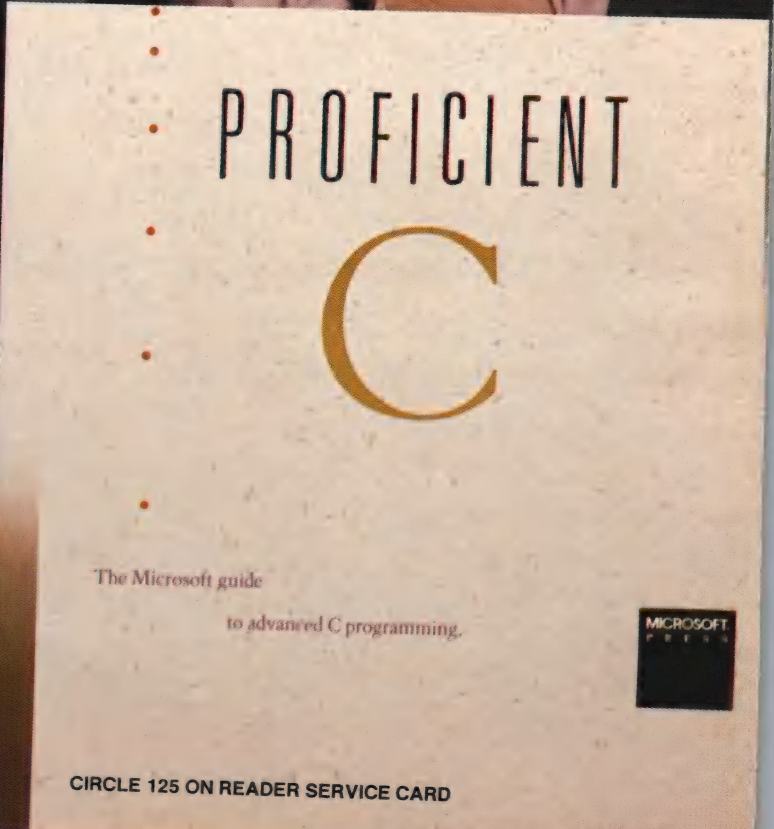
because the preprocessor intercepts the trailing comma and looks for a second macro argument. Note that the comma used in a *for* statement, such as *for(i=0, j=9; i<j ;)*, is indeed a comma operator.

Think of us as the Book of the Mind Club.









CIRCLE 125 ON READER SERVICE CARD

C CHEST

(continued from page 113)

downbeat; finally, it plays ten measures of 6/8. That is, you get a warning at the beginning of the measure that immediately precedes the new time signature.

This command can be abbreviated to *w* or *W* if you like (actually, any word that starts with a *w* will do). The frequency of the warning tone is not affected by the *–d* command-line switch, and it cannot be suppressed with parentheses.

Implementation

Click is implemented in Listings One–Seven, pages 82–96. Listing One, debug.h, is a general-purpose file that contains macros I use regularly. The *D( )* macro was discussed in the November 1986 C Chest (it's also in *Bound Volume 11*, page 740). Its argument goes away when *DEBUG* isn't defined. The *PUBLIC* and *PRIVATE* macros just declare a subroutine as static or not. They're useful because you can easily change the storage classes of all private subroutines to nonstatic when you're debugging. The *UX( )* and *MS( )* macros work just like the *D( )* macro does. Here the argument to *MS( )* is used only if MS-DOS is defined, and the argument to *UX( )* is used only if MS-DOS isn't defined (*UX* stands for Unix).

Listing Two presents hardware.h, a catchall file for IBM PC hardware defines. These macros define various numbers used for talking to the counter-timer chip in the PC and for turning the speaker on and off. The process is described in detail in the books listed in the bibliography. Timer channel 0 is used for the system clock interrupt, channel 1 triggers a memory refresh and shouldn't be touched by your program, and channel 3 is used to control the frequency of the PC's speaker.

The *SETFRQ* macro is interesting because it does more than one thing in a single macro. It's described in depth in this month's Flotsam and Jetsam. You pass it a desired frequency (in Hz), and it causes the bell to ring at that frequency the next time the speaker is enabled (with a *SPKR_ON( )* invocation). The frequencies of the 12 notes in the middle octave of an equal-tempered chromatic scale (as on a piano) are defined in notes.h (Listing Three) (C4 is middle C on a piano). Go up an octave by multiplying by 2, down by dividing. Go up a half step (from C to #C, for example) by adding the *TWELFTH_ROOT_OF_TWO* to a note.

The *beep(freq, duration)* subroutine (Listing Four) uses all this stuff to ring the bell on the PC at a specified frequency for an indicated number of seconds. Both of these numbers can be fractional. For example:

```
#define <notes.h>
beep( G4, 0.5 );
```

plays the G above middle C for a half second.

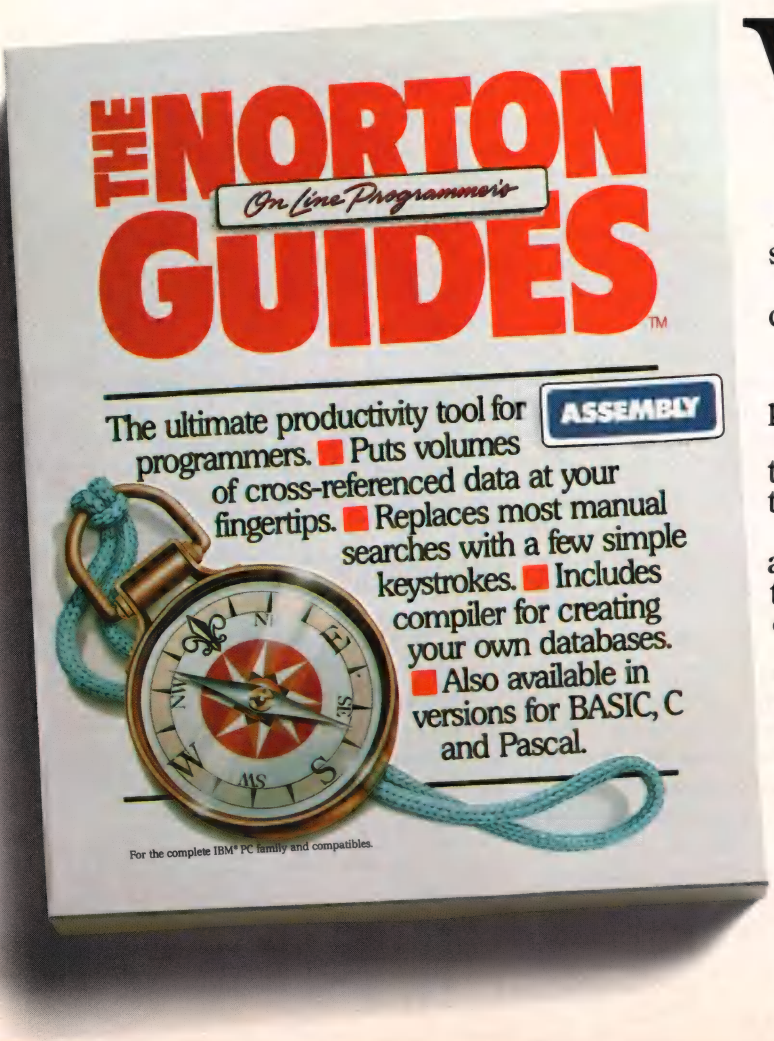
The ROM BIOS keeps a running count of clock ticks.

The note duration is determined by the argument sent to the *delay( )* subroutine, called on line 18 and declared in Listing Five. The ROM BIOS keeps a running count of clock ticks, accessible via function *0* of *interrupt 0x1a*. 0 is midnight on the day that the machine was started, and the interrupt returns with *AH* set to nonzero when you roll past midnight. *Delay( )* just waits in a tight loop for the correct number of ticks to pass.

Hitherto I've looked at pretty straightforward code, described in most books on DOS programming. With Listing Six, things get more interesting. The IBM PC system clock ticks once every 18.2 seconds (more or less). This resolution isn't adequate for musical applications—for example, click couldn't distinguish between metronome 150 and 152 at the default resolution. Moreover, sitting in a tight loop waiting for the clock to tick isn't a good way to do things—it's too easy to miss a tick.

These problems are solved in two ways. First, you have to speed up the system clock in such a way that you don't mess up the real DOS clock. Second, you have to provide the interrupt service routine to take care of the faster clock tick. If you speed up the clock by a factor of 4, for example, your interrupt service routine is

Peter Norton new programmi who hate



Nobody ever said programming PCs was supposed to be easy.

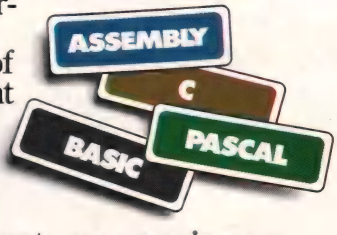
But does it have to be tedious and time-consuming, too?

Not any more.

Not since the arrival of the remarkable new program on the left.

Which is designed to save you most of the time you're currently spending searching through the books and manuals on the shelf above.

The Norton On-Line Programmer's Guides™ are a quartet of pop-up reference packages that do the same things in four different languages.



Each package consists of two parts: A memory-resident Instant Access™ program. And a comprehensive, cross-referenced database crammed with just about everything you need to know to program in your favorite language.

And when we say everything, we mean everything.

Everything from information about language

Designed for the IBM® PC, PC-AT and DOS compatibles. Available at most software

announces a ng tool for people manual labor.

syntax to a variety of tables, including ASCII characters, line drawing characters, error messages, memory usage maps, important data structures and more.

How much more?

Well, the databases for BASIC, C and Pascal give you detailed listings of all built-in and library functions.

While the Assembly database delivers a complete collection of DOS service calls, interrupts and ROM BIOS routines.

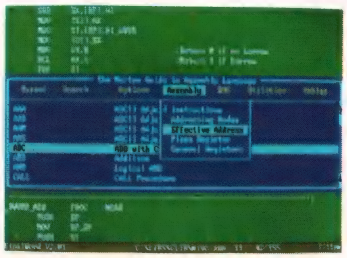
You can, of course, find most of this information in the books and manuals on our shelf.

But Peter Norton—who's written a few books himself—figured you'd rather have it on your screen.

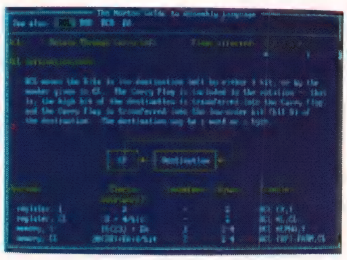
In seconds.

In full-screen or moveable half-screen mode.

Popping up right next to your work. Right where you need it.



A Guides reference summary screen (shown in blue) pops up on top of the program you're working on (shown in green).



Summary data expands on command into extensive detail. And you can select from a wide variety of information.

This, you're probably thinking, is precisely the kind of thinking that produced the classic Norton Utilities.™

And you're right.

But even Peter Norton can't think of everything.

Which is why there's a built-in compiler for creating databases of your own.

And why all Guides databases are compatible with the Instant Access program in your original package.

So you can add more languages without spending a lot more money.

To get more information, call your dealer. Or call Peter Norton at 1-800-451-0303 Ext. 40.

And ask for some guidance.

Peter Norton
COMPUTING

dealers, or direct from Peter Norton Computing, Inc., 2210 Wilshire Blvd., #186, Santa Monica, CA 90403. 213-453-2361, Fax 213-453-6398, MCI Mail: PNCI

CIRCLE 87 ON READER SERVICE CARD

(continued from page 115)

activated on every tick and your routine jumps to the default service routine on every fourth tick. As a final convenience, you want to be able to write your own interrupt service routine in C, without having to go to assembly language.

All of this is done by the routines in Listing Six. The *speedup( )* subroutine is called with:

```
speedup( factor, funct )
int     factor;
int     (*funct)( );
```

Factor is the speedup factor. Set it to 4 for a fourfold increase in the clock speed. The *funct* argument is a pointer to an interrupt service routine that is called on every clock tick. I'll look at one of these shortly.

Speedup( ) begins on line 93 of Listing Six. The first thing it does is remember the *funct* and *factor* arguments in two local variables: *service* and *tick_reset* (declared on lines 41 and 45). It also remembers the value of the current data segment (on line 103). You need to do this because it's convenient for the C interrupt service routine to access static data and global variables. Because the timer interrupt can come at any time, you have no idea what number is in the *DS* register when the interrupt is serviced; odds are it is incorrect (it will be the *DS* associated with the interrupted process). By remembering the *DS* register now, you can restore it later. Note that all the variables used by the interrupt service routine are being put into the *_TEXT* segment, which is normally used only for code. I'm doing this because the contents of the *CS* register must be valid when the service routine is entered (or you couldn't be there). So, you can always get at a variable in the *_TEXT* segment by using a *_TEXT:* or *CS:* segment override. Then you can retrieve the previously stored *DS* contents from a variable in the code segment.

When the C routine returns, the service routine decides whether or not to call the default timer interrupt.

Lines 108 to 124 of Listing Six speed up the clock hardware. Note that you have to test explicitly for a *factor* of 1 on lines 110 and 111 to avoid an arithmetic overflow exception. In this case, you want to load the counter with the default count of 0, which yields 64K ticks between interrupts (that's the default 18.2 seconds). A *speedup(1, . . .)* call is useful if you don't want to change the tick rate but do want to install your own service routine. Finally, the routine gets the old timer interrupt (8) vector, using DOS function *0x35* on line 133; saves it in *old_seg:old_off*; and then installs a new interrupt using DOS function *0x25* (on lines 133–140). You have to push the *DS* register (on line 136) because function *0x25* is passed the new vector in DS:DX. It's restored on line 140.

You'll note that I didn't actually install the user-supplied service routine in this last step; rather, I installed a pointer to the *serv* subroutine (lines 156–207). This routine sets up the machine environment so that the C routine can be called safely, and only then does it call the C function. *Serv* starts out by setting up a new stack (on lines 160–164). A small, 128-byte stack (declared on line 52) is used for this purpose. Then it pushes all the registers onto the new stack (lines 166–173). The old data segment is restored next so that the C function can get at global and local-static variables. Finally, the C routine is called (on line 179) indirectly through the pointer I set up earlier (on line 102).

When the C routine returns, the service routine decides whether or not to call the default timer interrupt. A running count (*numticks*) is decremented on each call. When it reaches 0, you reset it to *tick_reset* (the first argument to *speedup( )*) and then jump to the old service routine (on line 205), again indirectly through a pointer. If *numticks* hasn't gone to 0, you send a nonspecific EOI (end of in-

terrupt) to the hardware (on line 197) and then do an *iret* to terminate the interrupt.

The remainder of Listing Six is straightforward. *Slowdown( )*, defined on lines 213–247, just slows the clock down to its original rate and uninstalls the custom service routine. *Slowdown( )* must be called by your program before it terminates or the machine will go into outer space the next time a timer interrupt occurs. The *cli( )* and *sti( )* routines on lines 70–78 just disable and enable interrupts from C. They're useful when trying to avoid various communication problems between the interrupt service routine and the rest of the program. You'll see how in a moment.

Now that I've laid the groundwork, I can actually discuss the metronome program. Click.c is presented in Listing Seven. A few of the *#defines* at the top of the listing are interesting. *ROUND(x)* takes as input a floating-point number (either *float* or *double*) and converts it to an *int*. It rounds to the nearest integer value, however, rather than just truncating—the default behavior of the Microsoft compiler. The various definitions needed to figure the clock rates are on lines 42–46. *FACTOR* is the *speedup( )* factor. If the clock runs at less than 16 times the default tick rate, the program won't be able to resolve the timing by a single metronome count—it won't be able to differentiate between metronome 150 and 151, for example. *DEFAULT_TICK* is the default 18.2 times/second used by the system clock. *ONE_TICK* is the number of times that the clock ticks in a second, given the speedup factor defined earlier. Finally, *TICKS(x)* converts a metronome *x* into a number of clock ticks.

The virtual tape is defined with a system of *typedef*s on lines 60–78. Each measure is represented by the *MEASURE* structure defined on lines 63–72. *Num_beats* is the number of beats in the measure, and *ticks_per_beat* is the number of timer ticks between every beat. *Num_beats* is decremented on every beat, and when it reaches 0, the program goes to the next measure. When you're synchronizing with track 0 in a polyrhythmic mode, however, *num_beats * ticks_per_beat* is not necessarily the exact number of clock ticks in the corresponding measure. The *remainder* field makes up the difference. The extra ticks represented by *remainder* are spread over the first few beats of the measure to make up any discrepancy. This way the downbeats of synchronized measures will always coincide. *Cur_tick* is the current clock tick. It is initially set to *ticks_per_beat* and is decremented on every timer interrupt. When it reaches 0, a tone is output and the variable is reinitialized to *ticks_per_beat*. *Num_beats* is also decremented at this point. The rest of the structure is just housekeeping: *silent* is set when this measure is silent; *warning* is set when a warning pulse is to be used on the downbeat of each measure. All these fields are used (and modified) by the interrupt service routine on each clock tick.

A *TRACK* (on line 75) is an array of *MEASURE* structures, and the *Tape* (line 77) is an array of four *TRACK*s. The *Measure* array is an array of pointers, one for each track. These pointers point to the current measure on each track. They are incremented by the interrupt service routine every time it rolls over to a new measure.

The next set of global variables (lines 82–102) are used by the interrupt service routine to pass information to the main body of the program. *Ring_bell* is set when the bell should be sounded. It is set to one more than the track number or to the special value *WARNING*, defined on line 56, if the warning tone should be sounded. *Collision* is set true when there's a collision between two tracks (a beat occurs on both tracks at the same time). *Downbeat* is incremented on the downbeat of every measure on track 0; it's used to print the measure number to the screen. *Done* is set to true when all tracks are exhausted, and *Numticks* is incremented on each timer interrupt (it's useful primarily for debugging).

By far the largest subroutine in the program is *build_tracks( )* on lines 219–358, which parses the input file and initializes the *Tape[ ]* array. In spite of its length, it's pretty straightforward and requires little additional comment here.

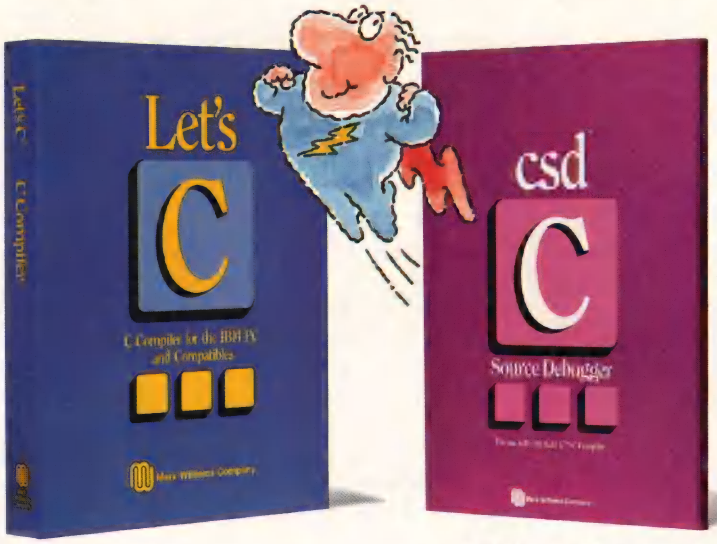
More important is the interrupt service routine itself, *timr_intr( )* on

PRESENTING THE DIFFERENCE BETWEEN FAST COMPILING AND FAST PROGRAMMING.

For compiling speed, you can't do better than Let's C. But to really speed up programming you can't do without the powerful source level debugger, *csd*.

If you want the power, portability and flexibility of C, start with the complete compiler, Let's C. For utilities, editor, compiling speed and fast, dense code, Let's C has it all.

But to get your programs up and running you need more. Because even the fastest compiler can't outrun bugs. You need the revolutionary C Source Debugger, *csd*.



CUT DEVELOPMENT TIME IN HALF WITH *csd*

csd lets you bypass the time consuming frustrations of debugging–like long dumps and clunky assembler. With *csd*, you actually debug in C. You learn faster because you watch your program run in C. You finish faster because *csd* combines the speed of a compiler with the interactive advantages of an interpreter. The end result? Development time is sliced in half.

LIMITED TIME OFFER FREE *csd* WITH LET'S C!

REVIEWERS ARE RAVING ABOUT LET'S C AND *csd*.

"Let's C is an inexpensive, high-quality programming package... with all the tools you will need to create applications."
–William G. Wong, *BYTE*, August 1986.

"The performance and documentation of the $75 Let's C compiler rival those of C compilers for the PC currently being sold for $500...highly recommended..."
–Marty Franz, *PC TECH JOURNAL*, August 1986.

"csd *is close to the ideal debugging environment...a definite aid to learning C and an indispensable tool for program development."*
–William G. Wong, *BYTE*, August 1986.

"This is a powerful and sophisticated debugger built on a well-designed, 'serious' compiler."
–Jonathon Sachs, *Micro/Systems Journal*, April, 1986

LET'S C AND *csd* FEATURES

NEW VERSION 4.0!

Let's C:

- Now compiles twice as fast
- Integrated edit-compile cycle: editor automatically points to errors
- Includes both small and large memory model
- Integrated environment or command line interface
- 8087 sensing and support
- Documentation features new lexicon format
- MS-DOS object compatible
- New make utility
- Fast compact code plus register variables
- Full Kernighan & Ritchie C and extensions
- Full UNIX compatibility and complete libraries
- Many powerful utilities including make, assembler, archiver, cc one-step compiling, egrep, pr, tail, wc
- MicroEMACS full screen editor with source included
- Supported by dozens of third party libraries
- For the IBM-PC and Compatibles
- Not copy protected

Sieve Benchmark

(Compile time in seconds)
Let's C: **2.8** (On 512K 6Mhz IBM-AT)
Turbo C: **3.89** (As advertised)

csd:

- Large and small memory model
- Debug in C source code, not assembler
- Monitor variables while tracing program
- Does not change program speed or size
- Provides separate source, evaluation, program and history windows
- On-line help screens
- Can interactively evaluate any C expression
- Can execute any C function in your program
- Trace back function
- Ability to set trace points
- Not copy protected

START TO FINISH, THERE'S NO BETTER ENVIRONMENT.

Get started with the right C compiler and you'll have everything you need for development–including source level debugging. On top of it all, Let's C and *csd* are today's best values in professional C programming tools. And most reliable: Mark Williams C compilers have been sold with DEC, Intel and Wang computers since 1981.

60 DAY MONEY BACK GUARANTEE

Mark Williams gives you a full 60 days to find out just how good Let's C and *csd* really are–or your money back.

So if you want more than a fast compiler–if you want your programs up and running fast, ask for Let's C and *csd*. You'll find them at your software dealer's, in the software department of your favorite bookstore, through the Express Program at over 5500 Radio Shacks or you can order now by calling **1-800-MWC-1700.***

*In Illinois call, 1-312-472-6659.

DDJ 987

Mark Williams Company

1430 West Wrightwood, Chicago, Illinois 60614

MARK WILLIAMS LET'S C AND *csd*. ONLY $75 EACH.

CIRCLE 102 ON READER SERVICE CARD

MICRO/SYSTEMS JOURNAL

FOR THE ADVANCED COMPUTER USER

SUBSCRIBE NOW AND SAVE OVER 15% OFF THE NEWSSTAND PRICE!

15% SAVINGS

Yes! I want to subscribe to Micro/Systems Journal™

and save over 15% off the cover price.

☐ 1 Year (6 issues) $20 ☐ 2 Years (12 issues) $35

☐ Please charge my: ☐ Visa ☐ MasterCard ☐ American Express

☐ Payment enclosed ☐ Bill me later

Card # _____ Exp. date _____

Signature _____

Name _____

Address _____

City _____ State _____ Zip _____

Savings based on a full one-year cover price of $23.70. Canada and Mexico add $3 for surface mail, $7 for airmail per year. All countries add $12 for airmail per year. All foreign subscriptions must be prepaid in U.S. dollars drawn on a U.S. bank. Please allow 6-8 weeks for delivery of first issue.

A Publication of M & T Publishing, Inc. 3020

15% SAVINGS

Yes! I want to subscribe to Micro/Systems Journal™

and save over 15% off the cover price.

☐ 1 Year (6 issues) $20 ☐ 2 Years (12 issues) $35

☐ Please charge my: ☐ Visa ☐ MasterCard ☐ American Express

☐ Payment enclosed ☐ Bill me later

Card # _____ Exp. date _____

Signature _____

Name _____

Address _____

City _____ State _____ Zip _____

Savings based on a full one-year cover price of $23.70. Canada and Mexico add $3 for surface mail, $7 for airmail per year. All countries add $12 for airmail per year. All foreign subscriptions must be prepaid in U.S. dollars drawn on a U.S. bank. Please allow 6-8 weeks for delivery of first issue.

A Publication of M & T Publishing, Inc. 3020

15% SAVINGS

Yes! I want to subscribe to Micro/Systems Journal™

and save over 15% off the cover price.

☐ 1 Year (6 issues) $20 ☐ 2 Years (12 issues) $35

☐ Please charge my: ☐ Visa ☐ MasterCard ☐ American Express

☐ Payment enclosed ☐ Bill me later

Card # _____ Exp. date _____

Signature _____

Name _____

Address _____

City _____ State _____ Zip _____

Savings based on a full one-year cover price of $23.70. Canada and Mexico add $3 for surface mail, $7 for airmail per year. All countries add $12 for airmail per year. All foreign subscriptions must be prepaid in U.S. dollars drawn on a U.S. bank. Please allow 6-8 weeks for delivery of first issue.

A Publication of M & T Publishing, Inc. 3020

No Postage
Necessary
If Mailed
In The
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT 790, REDWOOD CITY, CA

Postage Will Be Paid By Addressee

For the Advanced Computer User

Micro/Systems Journal

Box 3713
Escondido, CA 92025-9843

No Postage
Necessary
If Mailed
In The
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT 790, REDWOOD CITY, CA

Postage Will Be Paid By Addressee

For the Advanced Computer User

Micro/Systems Journal

Box 3713
Escondidio, CA 92025-9843

No Postage
Necessary
If Mailed
In The
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT 790, REDWOOD CITY, CA

Postage Will Be Paid By Addressee

For the Advanced Computer User

Micro/Systems Journal

Box 3713
Escondidio, CA 92025-9843

(continued from page 119)

lines 362–427. Though the routine is just a C subroutine, there are several issues that must be kept in mind while writing it. First, MS-DOS is not reentrant. In practice, this failing means that you can't call most DOS functions from an interrupt service routine (because the program might be in DOS when it was interrupted). The second issue is return values. Because the service routine isn't called in the normal way, it can't be passed parameters and it can't return a value in the normal way. Global variables must be used for this purpose. The remaining problems are stack-related. The interrupt service routine uses its own stack, so you have to disable the default stack-overflow checking that's inserted by the compiler (which will almost always fail because the stack isn't where its supposed to be). This disabling is often done with a compiler command-line switch, but the Microsoft compiler lets you do it with the *#pragma check_stack* directives (on lines 362 and 427). The *#pragma* lets you disable the checking for one subroutine only, instead of affecting the entire module. A trailing minus sign disables checking and a plus sign enables it. The final stack issue is its size. The service routine uses a 128-byte stack, of which 18 bytes are used to save registers and do the subroutine call. You have to be careful not only about the amount of stack used by your own local variables but also the amount of stack used by subroutines that the service routine might call. Be careful. If you need more stack, change the declaration for *stack* on line 52 of Listing Six.

The service routine itself modifies the global variables described earlier. The *for* loop is executed four times—one iteration for each track. The test on line 377 is for the end of track. If the beat count is 0, there are no more measures on this track. The next test (on line 379) checks to see if the bell should sound (this is the case if the current count is the maximum). The current tick is then decremented on line 402, and if it goes to 0, the number of beats is also decremented (on line 404). The routine advances to the next measure, if necessary, on line 409. The *else* clauses on lines 411–421 takes care of the *Remainder*—the extra clock ticks that have to be inserted to keep synchronized with track 0. I add 1 to the current tick count on line 418 to do this. That is, I stretch the current beat out by one clock tick. This way the extra ticks are spread over the first few beats in the measure instead of being piled in one place.

The *main( )* subroutine has to do several things to get the ball rolling. First, it must call *signal( )* to guarantee that *slowdown( )* is called if you abort the program with Ctrl-Break. *Signal( )* is called on line 495, and it installs the *on_break( )* subroutine (lines 431–442) to call *slowdown( )*. *Speedup( )* is called in the initialization part of the *for* loop on line 497. The loop terminates when the interrupt service routine says that it's finished by setting the *Done* flag. The code in the body of the loop just tests to see if it should ring the bell and rings it if necessary. I'm doing this here rather than in the service routine because it's easier. The problem is the delay between turning the bell on and then off again. You can't wait in the service routine itself because the timing would be thrown off. By waiting here, the wait time is essentially independent from the start-of-beat timing. Of course, you could turn the bell on in the service routine and then set a flag, decrementing the flag on each timer tick and turning the bell off when the flag gets to 0. This method seems a lot of bother, however, so I took the easy way out. The final point worth mentioning is that interrupts have to be disabled while you do the tests because there are two variables involved and you don't want one of these to magically change its value halfway through the test. Disabling interrupts prevents this change.

Availability

All the source code for articles in this issue is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal*, 501 Galveston Dr., Redwood City, CA 94063 or call (415) 336-3600, ext. 216. Please specify issue number and format (MS-DOS, Macintosh, Kaypro).

Because the code this month is pretty compiler dependent, I'm distributing executable versions (along with the full source code) through Software Engineering Consultants, P.O. Box 5679, Berkeley, CA 94705. This version has been enhanced to allow several different kinds of warning tones and to allow duration to be specified in minutes and seconds as well as beats. The cost is $20.

Bibliography

Brickner, Ralph G. "An Execution Profiler for the PC." *PC Tech Journal* 4:11 (November 1986): 120–142. This article describes another program that steals the system timer interrupt; the code of interest is in Listing 2, pages 140–142.

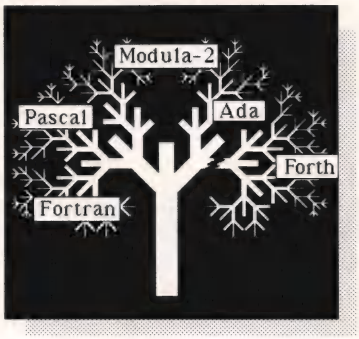
IBM Technical Reference. The BIOS listing contains the default timer interrupt service routine. It's on page 5-162 of the AT technical reference and on page A-79 of the XT reference. The routine is called *TIMER_INT* in both listings.

Norton, Peter. *The Peter Norton Programmer's Guide to the IBM PC*. Bellevue, Wash.: Microsoft Press, 1985. This book contains information on how the bell on the IBM PC works as well as information on the other DOS interrupts used by click. The information is duplicated in innumerable other books about programming the IBM PC.

DDJ

(Listings begin on page 82.)

V.I.P., Clustered Binary Trees, and Clustered List Data Structures



This month I discuss the Visual Interactive Programming (V.I.P.) language, a new icon-based interpreter for the Apple Macintosh computer. My second topic conforms with this issue's theme—algorithms—by presenting modified structures for binary trees and linked lists.

The Macintosh has been blessed with numerous language interpreters and compilers: BASIC, FORTRAN, Pascal, Modula-2, C, LISP, PROLOG, Forth, and so on. Now, Mainstay of Agoura Hills, California, has developed a new language that truly takes advantage of icons and the Macintosh user interface. V.I.P. is an interpreter that breathes life into a flowchart—instead of typing text for the source code, you can assemble a program using special flowchartlike symbols.

V.I.P. also incorporates a programming environment. Its appearance resembles MacPaint: a menu bar across the top; a flowchart viewing port; and to the left of this port, three groups of icons—object (data-type) icons, icons for loops and decision-making constructs, and icons for several classes of predefined routines. To build a program you point to an icon with the mouse and click. The environment's response takes one of two forms: a window or a flowchart icon. The window form is fairly typical of the Macintosh and involves a more sophisticated level of interaction with the user. The icon form lets you fully define the flowchart symbol and represents a simpler level of interaction. Each flowchart symbol has a comment line at the bottom.

by Namir Clement Shammas

The product uses the same geometric shape—namely, a horizontal rectangular strip—for all the icons. At the two edges of the rectangular icon are two squares: one to open the icon, the other to close it. Figure 1, page 123, shows a sketch of an opened FOR-NEXT icon. The rectangle containing the *for* keyword is the permanently visible part of the icon. The two columns below it are for input or inspection. The left column of rectangles labels the information required and encloses the object type in parentheses. V.I.P. uses lowercase and uppercase letters for the object-type codes to indicate input and output, respectively. The last row is reserved for comments.

V.I.P. supports a fixed set of object (data and constant) types. They are:

- *BYTE*, which uses 1 byte of storage. Short integers (–128 to +127) or single characters can use this type.
- *INTEGER*, which uses 4 bytes to accommodate long integers (between minus and plus 2 billion).
- *REAL*, which occupies 10 bytes with a 64-bit mantissa and 15-bit exponent. Thus, floating points with 19 significant figures are supported with an exponent varying from –4,932 to +4,932.
- *POINT*, which requires 4 bytes of storage to represent a vertical and horizontal set of coordinates. (The range of values for each axis is –32,678 to +32,767.) This is equivalent to a predefined record structure in a structured language such as Pascal or C.
- *RECTANGLE*, which uses 8 bytes to represent four integers that define the upper-left and lower-right corners of a rectangle.
- *CONSTANT*, which is used to implement symbolic constants. V.I.P. supports four types of constants: character, integer, real, and string. The *true*, *false*, *e*, and *pi* constants are predefined.

Arrays of up to three dimensions are supported, with the lower array bound fixed as 1. When you select a data-type icon, a window directory opens to display a list of all the variables of the selected type. The window lets you choose between global and local variables. You can also insert new variables, delete or change existing definitions, rename variables (V.I.P. is case sensitive), convert scalar ones into arrays and vice versa, and alter array sizes and the number of dimensions.

V.I.P. offers two decision-making constructs: *IF* and *CASE* statements. The *IF* statement comes in the form of the IF-THEN-ELSE icon, with two flowchart branches. You can simulate an *ELSEIF* by using nested *IF* statements. The IF icon is defined by entering a logical expression. The *CASE* statement supports up to 30 cases. When you choose the CASE icon, you are prompted for the number of cases to create an icon with the correct number of alternatives. You enter the selector (switch) variable and the values associated with each *CASE* clause. V.I.P. has no *CASE ELSE* clause.

The language supports two loop constructs: *FOR . . . NEXT* and *WHILE . . . DO*. When you use a *FOR* loop, you must specify the control variable as well as its initial, final, and step values. To insert a *WHILE* loop icon, you need to specify the logical expression used to iterate the loop.

V.I.P. lets you divide your task into smaller routines—a feature useful for maintaining a clear set of flowcharts instead of using one big flow-

chart. You can choose to invoke a routine's option from the top menu bar. This opens a new window in which you declare all your routines. You can easily switch between the main routine and any other routine to edit or inspect any program components. Each routine has its own workspace. When you select a routine for the first time, you start with a clear flowchart.

User-defined routines in V.I.P. can take parameters. A special window for parameters opens when you select the "Set argument . . ." option. The parameter window resembles the one for declaring variables, with a few differences. First, you must indicate the type and the input/output status. V.I.P. does not allow the same parameter to pass data back and forth between the routine and its caller. You can define the parameters as scalar or arrays (with up to three dimensions). Parameters can be inserted, deleted, or altered. V.I.P. maintains control over parameter passing and verifies that the arguments in the calls correspond to the parameters' type, number, and sequence.

V.I.P. has a versatile set of intrinsic functions. Mathematical functions include square root, logarithmic, trigonometric (and their inverses), and hyperbolic (and their inverses). Other mathematical functions include the absolute value, sign, modulus, minimum, maximum, and random-number generator. Another collection of functions returns the vertical and horizontal coordinates of a point and the four coordinates used to define a rectangle.

The software comes with powerful group of predefined routines. Each class of routines is represented by an icon located on the left-hand side of the flowchart viewing port. There are 15 classes of routines, listed along with their functions in Table 1, right.

The V.I.P. editor enables to you perform search/replace operations on the contents of the flowchart icons and also lets you cut, copy, paste, and delete icons. You can view the flowchart using a smaller scale that can be magnified by pressing the shift key and the mouse button simultaneously. In this mode, however, the screen displays uncommented icons, which I find annoying.

Finally, V.I.P. includes a versatile debugger that lets you single-step through your flowchart icons, set and remove breakpoints, and set/examine objects.

I ran some of the V.I.P. demonstration programs, which illustrate how easy it is to write programs that use the Macintosh interface. I also wrote two versions of the Sieve benchmark program. The first (see the text version in Example 1, page 124) is written in a typical form—that is, no local routines are used. One iteration of this program took 3 minutes and 35 seconds. Example 2, page 125, shows the text of the second version, which uses two subroutines—*init* and *body*.

To the best of my knowledge, V.I.P.

1. Assignment: includes simple assignment and filling/copying bytes.
2. Mathematics: includes integer, fraction, power, simple financial calculations, and sorting numbers.
3. String manipulation: to carry out typical related operations (string concatenation, comparison, append, copying, and conversions with numbers). Sorting an array of strings is also included.
4. Graphics: a large set of routines that lets you obtain the best of the Macintosh graphics.
5. Event trapping: to inspect the status of the mouse.
6. Menu management: to create, enable, disable, remove, and load menus, to name a few.
7. Window management: to create, set up, load, remove, and activate windows.
8. Text editing: to perform text editing functions, such as copying, pasting, cutting, clearing, and inserting text. Text file I/O operations are also included.
9. Dialog management: to permit your programs to display Macintosh dialog windows.
10. Sound effects: to play tune, set voice, set notes, and turn sound on/off.
11. Record management: to support dynamic record allocation, copying of records, and field I/O.
12. I/O operations: a set of I/O routines for file manipulation (open, close, get file position, and so on) and I/O of objects, records, text, and pictures.
13. Printing: to set up the printed page and print text and pictures.
14. Branching: to exit from loops and execute another program.
15. Date and time management: to wait, read the clock, and obtain the time and the date.

Table 1: *V.I.P.'s classes of routines and their functions*

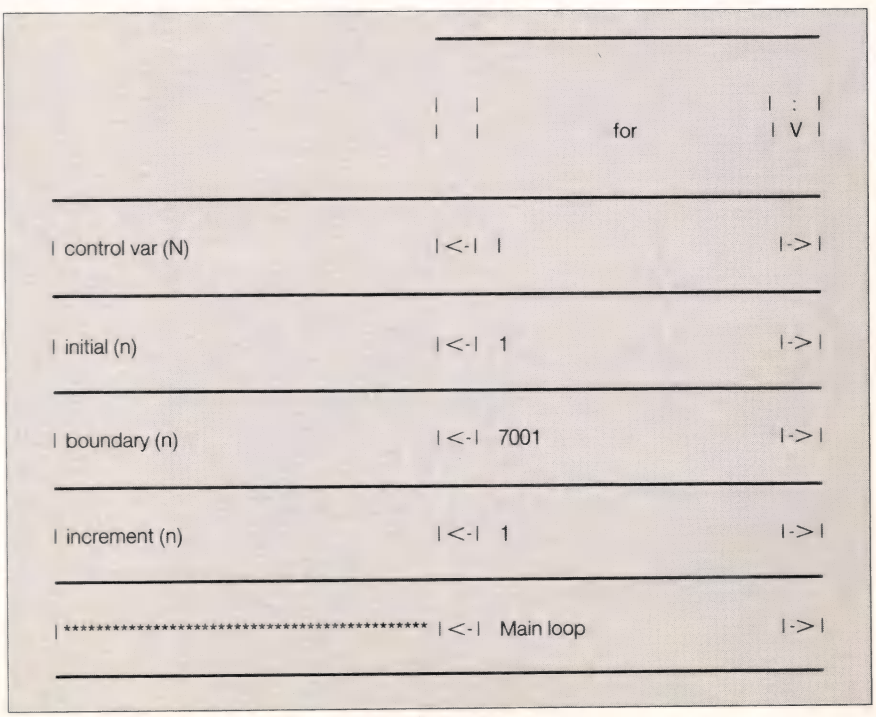


Figure 1: *Sketch of an opened FOR-NEXT icon*

(continued from page 123)

is the first commercially distributed program of its kind for microcomputers. It offers several interesting programming features—in particular, visual programming, which is excellent for teaching because students don't look at dummy flowcharts but at active ones. Visual programming may also prove valuable for algorithm design. In addition, V.I.P.'s library of routines make it a very capable language. I applaud Mainstay for its efforts and creativity in developing V.I.P. and look forward to more powerful versions. Mainstay has also announced that it will be realeasing V.I.P. translators for Pascal and C.

I have praised both V.I.P. and the concept of visual programming; now let me mention a few shortcomings. First, programmers who type quickly and use keyboard macros may find software development in visual programming languages slow. Second, V.I.P. does not provide an escape mechanism once a flowchart icon is opened. Third, viewing large flowcharts is cumbersome. Fourth, programmers have to inspect V.I.P.'s flowchart icons in order to view their contents. And finally, V.I.P. makes no provision for user-defined objects.

Clustered Binary Trees

Binary trees are useful data structures for internal sorting and searching. Yet, despite all the praise binary trees receive, they are vulnerable to becoming unbalanced. The ultimate nightmare is to input a perfectly sorted array into a binary tree and end up with one long linked list! Two decades ago, two Russian mathematicians devised a few algorithms to maintain a binary tree in a near-perfect balanced state. This type of binary tree is known as an AVL tree. Yet both the basic binary tree and the AVL tree typically use less-than comparisons to insert a new node or to search for one—they take no advantage of the difference between the values of a resident node and an incoming data item. With binary and AVL trees, you compare the value of a data item with that of a node. If it is less or equal, you use the left node pointer; otherwise, you use the right node pointer.

I suggest the following modification to the binary tree. In the new tree, each node has two left pointers and two right pointers, and so I call it the clustered binary tree (CB tree for short) and name the pointers low left, high left, low right, and high right. Why duplicate the pointers for each side? The answer lies in being able to select the proper pointer to follow, based on the difference between the values of the node key and the incoming data item. A critical difference value (CDV) is preassigned based on the nature of the data. When handling numeric keys, the values of the CDV and the whole algorithm are easy to implement. If the calculated difference is greater than the value of the CDV, then the high-left or high-right pointers are used, depending on the sign of the difference; otherwise, the low pointers are used.

The use of these four pointers in the CB tree helps to separate nodes with keys that vary widely and so speeds up tree insertion and searching. I used the Turbo Pascal program shown in Listing One, page 98, to compare the speed of insertion and searching in binary and CB trees. The program creates an array of numbers by using one of three methods: random-number generation, the sine function, or the cosine function. Although the speed of searching was about the same, the CB tree was four to five times faster in inserting new data. I used the sine and cosine func-

```
byte

Ch


integer

Count
Diff
Flag[7001]
I
Iter
K
Prime
T1
T2


constants

SIZE = 7001


main

read clock (T1)
for (Iter,1,1,1)  ** Main loop **
        assign (0,Count)
        for (I,1,SIZE,1)  ** Init. flags **
                assign (1,Flag[I])
        for (I,1,SIZE,1)
                if (Flag[I] = 1)
                        assign (I+I+3,Prime)
                        assign (I + Prime,K)
                        while (K  SIZE)
                                assign (0,Flag[K])
                                assign (K+Prime,K)
                        assign (Count+1,Count)
                else

read clock (T2)
wait (5000)
assign ((T2-T1)/60,Diff)
write output (1,"@i",Diff)
get key (100,Ch)
```

Example 1: *V.I.P. text output for the first version of the Sieve program*

tions to generate arrays with a certain degree of order, which is the weak point of the binary tree.

If string-type keys are used with CB trees, then the ASCII code numbers of the first few leading characters are utilized. You use the Pascal *ORD( )* function to do this. The numeric values used for critical difference calculations are *ORD(Key[1])* for using the first character in the key, and:

ORD(Key[1]) * 100 + ORD(Key(2))

for using the first two characters.

Clustered List Structures

You can also use the concept of the clustered binary tree structure with lists. Basically, a clustered single-link list (C list for short) is a list with two sets of pointers: one set forms the "high-track," or fast search lane; the other forms the "low track," or a clustered sublist. Thus, a C list structure has one high-track link and many clustered sublists, each linked to a high-track node, and so a high-track node becomes an index that indicates the range of data stored in its linked sublist. The effect of the two sets of pointers is best visualized by the two lanes of a highway, in which one is for faster traffic. You normally take the fast lane, as long as you are relatively far from the exit you seek. As you get closer to your exit, you switch to the slower lane. The same idea applies to C lists. By using numerical differences in comparing a node with an incoming datum, you can decide whether or not to use the high-track pointers and so bypass many unnecessary comparisons. This approach makes C lists more efficient in searches than are normal lists. The price you pay is the extra set of pointers.

Listing Two, page 102, shows a Turbo Pascal program that demonstrates C list insertions and displays. The program contains procedures for searching, inserting, and visiting C lists. Notice the following aspects of the program:

1. A string-type key is used. The entire key of any node and incoming data is used in a logical comparison. The numeric difference is calculated for the first character only, however.
2. A CDV of 0 is used, which causes the high-track node to index on a range of one character. The range of characters is (CDV+1) when the ASCII number of one key character is used.
3. The *Search_Node* procedure uses a Boolean flag, *HiTrack*, to switch from using high-track pointers to

```
byte

Ch


integer

Count
Diff
Flag[7001]
Iter
T1
T2


main

read clock (T1)
for (Iter,1,1,1)
        init  ** Initialize flags **
        body (Count)  ** Body of sieve **
read clock (T2)
assign (T2 - T1,Diff)
write output (1,"@i",Diff)
get key (5000,Ch)


body (Count)


<-- integer Count


integer

Flags[7001]
I
K
Prime


assign (0,Count)
for (I,1,7001,1)
        if (Flag[I] = 1)
                assign (I+I+3,Prime)
                assign (I+Prime,K)
                while (K  7001)
                        assign (0,Flag[K])
                        assign (K+Prime,K)
                assign (Count+1,Count)
        else


init

integer

I


for (I,1,7001,1)
        assign (1,Flag[I])
```

Example 2: *V.I.P. text output for the modified Sieve program*

low-track pointers during a search.
4. A new datum can be inserted in one of the following locations:
- as the new head of the entire list
- in a clustered sublist
- as a new high-track node
- as the new member of a high-track node, pushing the previous one inside the clustered sublist

The unused high pointers in a sublist node can be used to form a doubly linked sublist. I feel the impact of C lists on list structures is greater than that of CB trees on binary trees. The improved performance brought by C lists is less affected by the type and variation of data than is the case with CB trees.

Availability

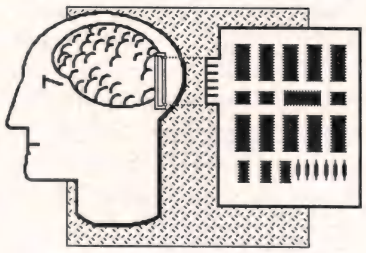
All the source code for articles in this issue is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal*, 501 Galveston Dr., Redwood City, CA 94063 or call (415) 366-3600, ext. 216. Please specify the issue number and format (MS-DOS, Macintosh, Kaypro).

DDJ

(Listings begin on page 98.)

Vote for your favorite feature/article.
Circle Reader Service **No. 6.**

Smalltalk/V



Digitalk's Smalltalk/V programming tool is a bit-mapped implementation of a substantial subset of Smalltalk. It is aimed primarily at the AI development market and is code compatible with the earlier Methods product, also by Digitalk. The feature that makes it suitable for AI, aside from the object orientation, is primarily the inclusion of a surprisingly complete and robust PROLOG compiler. This PROLOG includes many predicates that are lacking in Turbo Prolog, for example, such as *functor* and *univ*.

Smalltalk/V also has excellent graphics capabilities, such as turtle graphics, and offers good performance in graphics animation. Also included with the product is a large on-disk tutorial that provides some substantial program examples. Digitalk's Methods was the first Smalltalk implementation for PCs, and it was an important landmark as far as many programmers were concerned. But here we have a major subset of it running in about half a megabyte.

The first thing to realize about Smalltalk/V is that it is not just a programming language, but as any Smalltalk should be, it is a full, multiwindow environment with drop-down menus, mouse support, and more conveniences than you have probably ever seen in a programming environment on a PC. A

by Ernest R. Tello

mouse is not necessarily required because Digitalk has implemented an ingenious use of the keyboard using two main clusters of keys on the far left and far right, which it calls the left-hand mouse and right-hand mouse, respectively. If you choose not to use a mouse, you will find that after a while this works quite well. You can keep both hands in their place and reach for one of the keys in either cluster just as if you were pressing the buttons on two mice. The product was intended to be used with a mouse, however, and works much better with one, if for no other reason than that it makes the cursor fly across the screen instead of crawl. At this time, Smalltalk/V supports the Microsoft mouse and the Mouse Systems mouse. I have also had no trouble using it with the Logitech mouse.

The current version of Smalltalk/V comes on three disks—the Image, Source, and Tutorial disks—and requires 512K. It is a considerable accomplishment to have implemented so much of Smalltalk-80 in half a megabyte.

The dialect of Smalltalk/V is so close to Smalltalk-80 that most of the classes and examples in the Smalltalk-80 book series can be entered "as is." The main exceptions are those that make use of multitasking, such as the simulation examples—the system accepts even these, although they won't work as written. This is important for programmers new to Smalltalk because there is really very little published material available other than what is available for Smalltalk-80 to give them a full overview of the Smalltalk system and help them get going with it.

In the Class Hierarchy Browser, the lower classes in the hierarchy beneath those that are the immediate subclasses of *Object*, the root class, can be either hidden or visible as you choose. Once you have chosen a particular class with subclasses, you can choose to show or conceal just the subclasses under it. Smalltalk/V also has some additional commands on the desktop—for example, now you can cycle around to other windows from a command on one of the drop-down menus. This was needed because, when a window is completely covered by another window, you cannot select it with the mouse.

The basic types of facilities you use with Smalltalk/V are things such as workspace windows, browsers, menus, and occasionally what are known as inspectors. The main types of browsers are Class Browsers, Class Hierarchy Browsers, and the Disk Browser. These are specialized window facilities that give you a viewpoint onto a particular aspect of the system. And as I mentioned earlier, you can create as many instances of these views as you may need.

Class Consciousness

A Class Hierarchy Browser is the Digitalk version of the Smalltalk System Browser. As implemented in Smalltalk/V, this type of browser has four separate panels. The first is a scrolling window that lists the main classes and subclasses in the system. To the right of it is the methods pane, which displays the list of applicable methods. Beneath it are two small selector panes containing the words *class* and *instance*, respectively. Finally, on the bottom is a large pane that displays the actual Smalltalk source code for selected items. Depending upon whether you select on the instance or class pane, either the calling names of instance or class methods are displayed in the methods pane. When you select one of these method names, its source code is displayed in the lower pane. When the source pane is current, it acts as a

What experts are saying about PC Scheme from Texas Instruments:

"Texas Instruments has produced an outstanding implementation of a language that could well become as indispensable as C is today."

AI Expert, February 1987

Discover how powerful—and inexpensive—PC symbolic programming can be with PC Scheme from Texas Instruments. Whether you're an experienced Lisp programmer or just beginning, PC Scheme is the complete, $95* solution to your software development needs.

PC Scheme combines elegant simplicity with remarkable speed in a full Lisp development system. Named *PC Tech Journal*'s Product of the Month (August 1986), PC Scheme brings professional Lisp programming features to personal computers.

PC Scheme 3.0

- —Optimizing incremental byte-code compiler for ease of programming and operation
- —EMACS-like editor
- —Lexical scoping of variables
- —Ability to suspend PC Scheme, execute DOS-based programs, then return to PC Scheme
- —Random-file access and binary-file support
- —Extensions for debugging, graphics and windowing
- —External language interface to C, Turbo Pascal® and other languages
- —SCOOPS (Scheme Object-Oriented Programming System)
- —Two-megabyte extended/expanded memory support
- —New manuals with tutorials and examples

Find out for yourself why experts are praising PC Scheme. For the dealer nearest you, or to order by phone, call toll-free:

1-800-527-3500

* *TI Suggested list price*

PC Scheme runs on IBM® Personal Computers and compatibles (including the Texas Instruments Business-Pro™ computer). Minimum configuration: 512K RAM, dual floppy system.

Turbo Pascal is a registered trademark of Borland International. IBM is a registered trademark of International Business Machines Corporation. Business-Pro is a trademark of Texas Instruments Incorporated.

TEXAS INSTRUMENTS

261933C

(continued from page 128)

text editing pane in which you can create and modify source code.

What this type of browser means to a software developer is that you have a built-in overview of the system that can give you ready access to anything in the system at all times. The way you generally call upon things that have already been entered into the system is by creating instances of a class and initializing variables in a workspace window and then sending messages to it. Any involved interaction can itself be made into a program by adding it as a new subclass with its own variables and methods that can be instantiated and evaluated more easily.

As mentioned earlier, Smalltalk/V also provides inspectors. These are special-purpose windows you can use as low-level debugging tools that allow you to examine and even change objects in the system. You don't open an Inspector window by accessing a menu; instead, you use a workspace or system transcript window to send the inspect message to an instantiated object. An Inspector window with two panes—one on the right and one on the left—then opens. The pane on the left displays the names of the instance variables, and the right pane shows their values.

Two fonts come with Smalltalk/V. The fonts differ mainly in size—one has 8 × 8 pixels and the other 8 × 14 pixels. Other features included in Smalltalk/V that were absent in Methods are a DOS shell, a garbage collector, and virtual memory management.

The Smalltalk/V manual is the *Tutorial and Programming Handbook.* In many respects it is remarkably clear and well written. Its main shortcoming is that, in spite of its thoroughness and readability, it is not a complete reference to the behavior of the Smalltalk/V system. I would like to see a companion reference guide that would go more deeply into the behavior and implementation of the garbage collector and virtual memory, for example.

Graphics

An interesting approach to graphics has been adopted in Smalltalk/V. The basic class that implements the graphics capability is the *BitBlt* class, which is named for the bit block transfer operation and is much as in Smalltalk-80. Together with its immediate subclasses *Pen* and *CharacterScanner* and the subclasses of *Pen – Commander* and *Animation – BitBlt* provides the basis for how Smalltalk/V creates bit-mapped displays. The block transfers occur between two *Forms*—a source *Form* and a destination *Form*. *Forms*, *Points*, and *Rectangles* constitute the main structures used in Smalltalk/V graphics. A mechanism called a clipping rectangle is used—again, much as in Smalltalk-80—to define the maximum size of the bit transfer. This clipping rectangle restricts the size of the rectangular array of bits that will constitute the destination *Form*.

To see how this works, first look at the section of the class hierarchy in question:

```
BitBlt
  CharacterScanner
  Pen
    Animation
    Commander
```

The *CharacterScanner* class has the

job of converting ASCII character codes into displayable bit patterns. The *Pen* class, as you might have surmised, is the class that implements turtle graphics. The *Animation* class constitutes collections of pens that represent the various objects being animated. Finally, the *Commander* class controls arrays of pens in such a way that, whenever it receives pen-related messages, it relays the same message to each of the pens under its command.

It is also interesting to see how Smalltalk/V handles windows. Here, the approach is very different from that used in Smalltalk-80 but is essentially the same as that used in Methods. The following segment of the class hierarchy comes into play:

```
Dispatcher
  GraphDispatcher
  PointDispatcher
  ScreenDispatcher
  ScrollDispatcher
    FormEditor
      ListSelector
    TextEditor
      PromptEditor
  TopDispatcher
DispatchManager
```

Browsing Drives

The Disk Browser is one of the more original and powerful facilities in the Smalltalk/V system. Its window is composed of four panes. In the upper-left pane is the directory hierarchy list, which shows all the directories on a disk. In the pane to the right of this is the file list, which displays the names of all the files in the selected directory. A large pane below these is the contents pane, which displays either the screen of directory information as it would appear after an MS-DOS *dir* command or the contents of a selected file. And just above this, there is a small pane called the directory order pane, which displays the way directory information is currently being sorted for display in the contents frame—such as by date, name, or size. With this facility you can create or remove directories and files as well as rename, copy, or print them.

With the command menus accessible from the contents pane, you can perform just about any file maintenance operation, including cut and

(continued from page 131)

paste, copy, read, and install. The full set of commands available as displayed in the menu varies depending upon the size of the file. Normally, with files greater than 6,000 bytes, the contents pane displays only the first and last 2,000 bytes. In that event, you can use the *read it* command to read in the complete contents of a large file. Also available is the *save as* command, which allows a file to be saved under a different name. Finally, the *install* command allows source files to be compiled into the Smalltalk/V system.

One thing you have to keep track of is the size of the changes.log file. If it starts to get large, then there is a facility for compressing it. You must use this before the changes log gets too large and space on the disk runs low, or your image will die the death. The log facility is an essential one for those who can't resist taking advantage of the fact that Smalltalk is internally extensible to a large degree, as are Forth and LISP. While modifying the internals to create an image of a new dialect of Smalltalk/V, prior to getting your modified system debugged, it is bound to crash more than twice. The log file is insurance that you will never lose anything you've done for keeps, unless for some reason a crash clobbers this file. If necessary, you can even use it to restore the system image.

A Method to the Madness

At the very center of all this are the methods—the actual modular subroutines that do the message passing. There are two different types of methods—instance methods and class methods, which are analogous to the instance and class variables.

One important departure of Smalltalk/V from the Smalltalk-80 standard is the omission of the *ClassDescription* class. In Smalltalk-80, the class hierarchy starting with the *Behavior* class is organized like this:

```
Behavior
  ClassDescription
    Class
    Metaclass
```

The arrangement in Smalltalk/V is the same except that *ClassDescription* is omitted. As a result, Smalltalk/V does not support message categories—that is, the grouping of methods for a given class under various category names. In many cases, I have found it relatively easy to add missing Smalltalk-80 classes to Smalltalk/V. In this case an addition is not easy to make because, if *ClassDescription* is added as a subclass of *Behavior*, it becomes a peer class of *Class* and *Metaclass* rather than a superclass of them.

On the other hand, Smalltalk/V has several special classes that are not found in Smalltalk-80. Table 1, below, contains a list of those that are not simply machine-specific classes but have to do with the IBM PC implementation and user interface.

Example: A Small Inference Engine

To give you a better grasp of specific things that Smalltalk/V can do, I'll now discuss an example of a simple rule-based reasoning system that was provided by Digitalk. First, let's look at its overall structure.

There is a kind of application holder class called *InferenceEngine* that has three subclasses—*Expert*, *Fact*, and *Rule*—that do all the work. An instance of the class *Expert* is a working inference engine that can evaluate rulebases for particular subject areas or domains. In this case it is a tree expert. The inference engine is a simple forward-chaining one that accepts a set of facts and applies the rules to the facts it already knows to determine if the goal is true. It then allows you to ask it to explain how it

```
CursorManager
Directory
DiskBrowser
Dispatcher
DisplayString
FixedSizeCollection
IndexedCollection
InfiniteForm
LinkedListStream
MethodDictionary
StringBlt
StringModel
SystemDictionary
```

Table 1: *Classes unique to Smalltalk/V*

HOT GRAPHICS PACKAGE FOR C PROGRAMS.* $39.95

Everything you need to write dramatic graphics effects into your Eco-C88 C programs. Some of the features include:

- Support for EGA, CGA, and Z100
- Over 100 graphics and support functions, many of which are PLOT-10 compatible.
- Many low level support routines reside outside your small model code-data area
- Can write dots thru the BIOS (for compatibility) or to memory (for speed)
- Graphics function help from CED editor available
- World, pixel or turtle color graphics modes
- 47 standard fill patterns, 17 line dashing patterns, Hershey fonts, plus user defineable fill, dash and fonts
- Supports view areas, rotateable fonts, clipping, arbitrary fill areas, extensive error checking, examples, and user's manual.

A must for the graphics enthusiast and a bargain at only $39.95

***Requires Eco-C88 C Compiler.**

NEW POP-UP WINDOWS FOR YOUR C PROGRAMS.

This windowing library allows you to add pop up windows in your C programs quickly and easily. Use them for help windows, selection menus, error messages, special effects—anywhere you need an attention getter. Just some of the features include:

- CGA, EGA, and monochrome support
- Slow mode option for "flicker" displays
- Control any program that goes through the BIOS
- Use up to 255 windows
- No special window commands; use print f ()
- Resize and move windows
- Custom window titles and borders
- Can be used with ANSI device driver
- Most of window's code-data lies outside small model limits
- Use any of the IBM text or block characters
- User's manual and examples

The Windowing Library requires an IBM PC compatible BIOS and the Eco-C88 C compiler.

ONLY $29.95

HANDY LIBRARIAN MAKES LIFE EASIER.

Now you can combine your modules, functions, and subroutines into your own library for easy link commands. Fully compatible with ANY standard OBJ format files (not just Ecosoft's products). With the Ecosoft librarian, you can:

- Add, delete, and extract from a library
- Get table of contents or index of a library
- Combine libraries, control library page size, use switches for combinations, process complex library requests, use wildcards, and do library directives from command files.
- Complete with user's manual

A valuable addition for any programmer.

ONLY $29.95

THE FIRST PROFESSIONAL 'C' COMPILER FOR UNDER $60.

A C compiler with many ANSI enhancements at an unbelievably low price. The Eco-C88 C compiler has:

- Prototyping (the new type-checking enhancement)
- Enum and void data types
- Structure passing and assignment
- All operators and data types
- A standard library with more than 200 functions (many of which are System V compatible for greater code portability)
- CC and mini-make that all but automates the compile process
- 8087 support (we sense the 8087 at runtime — no dual libraries)
- ASM or OBJ output for use with MSDOS linker
- Tiered error messages — enable-disable lint-like error checking
- Fast compiles and executing code
- Expanded user's manual
- CED full-screen program editor

Everything you need at the unbelievable price of $59.95.

Eco-C88 C compiler requires an IBM PC, XT, or AT (or compatible) with 256K of memory, 2 disk drives and MSDOS 2.1 or later.

Orders only:
1-800-952-0472
Technical Information:
(317) 255-6476

NOT COPY PROTECTED.

Ecosoft Inc.
6413 N. College Ave.
Indianapolis, IN 46220

ORDER FORM CLIP & MAIL TO: Ecosoft Inc., 6413 N. College Ave., Indianapolis, IN 46220

ITEM	PRICE	QTY	TOTAL
Flexi-Graph Graphics	$39.95		
Window Library	$29.95		
Eco-Lib Librarian	$29.95		
Eco-C88 C Compiler CED	$59.95		
		SHIPPING	
		TOTAL (IND. RES. ADD 5% TAX)	

PAYMENT: ☐ VISA ☐ MC ☐ AE ☐ CHECK

CARD # _____ EXPIR DATE _____

NAME _____

ADDRESS _____

CITY _____ STATE _____

ZIP _____ PHONE _____

CIRCLE 89 ON READER SERVICE CARD

ECOSOFT

(continued from page 132)

arrived at its result. The main operation of the inference engine is to apply rules to test a result. It sends a message to the rules collection to test the facts and then applies the *eval:* method to see if the goal to be tested matches. If so, it returns true; if not, it returns false.

The *action* method fires the conclusion of a rule and in doing so adds a new fact to the facts collection. The way you use this little demo expert system is by entering everything it needs directly in a text area and then causing the interpreter to evaluate it by selecting DO IT from the drop-down menu. This does the following things: First, it creates an instance of the *Expert* class called by the global variable name *Tree*; second, it initializes an instance of the *Fact* class; and third, it submits various facts about a tree for evaluation by adding them to the *Fact* collection. This sends the message to the *Tree* expert to prove the goal that, given the previously entered facts, for example, the tree is of the cyprus family.

Several things should be noted about this inference engine demo, which was not intended to be anything more than a simple toy demonstration. First, it does not use a rule syntax other than Smalltalk itself, so no parser is needed. This allows it to run quickly but makes the rules less readable—a familiar trade-off. Another point is that it lacks the ability to read in the fact and rule collections directly from a file—a facility that could be added without too much difficulty. There is also no real user interface other than the *explain* method. In particular, there is no facility for posing questions to users about values that the system cannot find otherwise. The point is that, although this is a toy system, it points to what a full system might be. Most of these features are not too difficult to add, once you grasp the basics of how to implement an inference engine in Smalltalk.

The advantages of implementing a full expert system shell in Smalltalk/V are quite easy to see. First, you get the lush, easy-to-understand user interface practically for free. It would not be a major development project to use the Smalltalk/V menu and windowing facilities to build a superior expert system consultation environment. Another important plus is the multiple instance aspect of Smalltalk. You can have as many *Experts*, like the *Tree* expert, initialized as necessary, each with a separate ruleset. Smalltalk/V could then be programmed for one *Expert* to pass a message to another for a goal to be tested. Also, more flexible inference methods could be implemented for backward chaining and combining both forward and backward chaining. Finally, a parser could be written that could accept a more "friendly" rule syntax and compile it into the Smalltalk format as used here for running finished knowledge bases.

Conclusions

Smalltalk/V, Version 1.2, is a remarkable accomplishment and a very easy-to-understand environment for newcomers to Smalltalk to get acquainted with the language. Its performance is surprisingly fast, considering all the things going on.

I should also mention that the Goodies Extension Kit disk, which is now available as an option for Smalltalk/V, contains an implementation of multiprocessing that could be important if discrete simulation is what you're after in a Smalltalk environment. The simulation examples in the Xerox Smalltalk series presume the multiprocessing capability of Smalltalk-80. Now, with the multiprocessing classes provided on the optional Goodies disk, this should not be a problem.

DDJ

Vote for your favorite feature/article.
Circle Reader Service **No. 7.**

Vendor

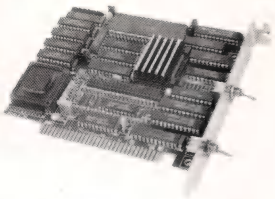
Smalltalk/V
Digitalk Inc.
5200 Century Blvd.
Los Angeles, CA 90045
(213) 645-1082
Reader Service No. 127

MICROWAY ACCELERATES YOUR PC!

FastCACHE-286™

Runs your PC Faster than an AT!

Runs the 80286 at 9 or 12 MHz and the 80287 at 8, 9 or 12 MHz. Includes 8 kbytes of 55ns CACHE.



Compatible with IBM PC, XT, Leading Edge Model D, Compaq, and Turbo motherboards. Includes 8088 Reboot Switch, DCache, Print Spooler and Diagnostics ..... **From $399**

LOTUS/INTEL EMS SPECIFICATION BOARDS

MegaPage™ The only EMS board which comes populated with two megabytes of cool-running, low power drain CMOS RAM installed. Includes RAM disk, print spooler, disk cache and EMS drivers. For the IBM PC, XT and compatibles... **$549**

MegaPage with ØK **$149**

MegaPage with 2 megabytes of HMOS RAM **$419**

MegaPage AT/ECC™ EMS card for the PC AT and compatibles includes Error Correction Circuitry. With ECC, 11 RAM chips cover 256K so the user never encounters RAM errors. With 1 megabyte CMOS RAM **$699**

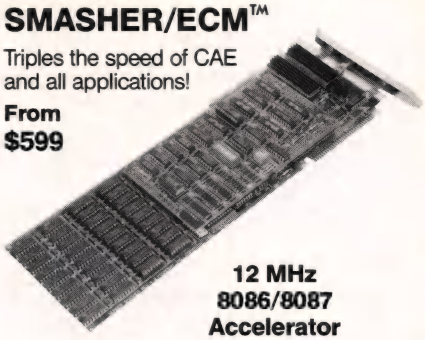
INTEL, JRAM, or Maynard .... **CALL**

INTEL INBOARD 386 ØK **$1250**

NUMBER SMASHER/ECM™

Triples the speed of CAE and all applications!

From $599



12 MHz 8086/8087 Accelerator Plus A Megabyte for DOS!

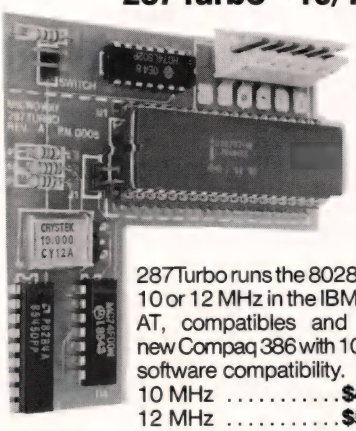
For the IBM PC, XT and compatibles

PC Magazine **"Editor's Choice"**

8087 SOFTWARE

IBM BASIC COMPILER	**$465**
MICROSOFT QUICK BASIC	**$79**
87BASIC COMPILER PATCH	**$150**
87BASIC/INLINE	**$200**
IBM MACRO ASSEMBLER	**$155**
MS MACRO ASSEMBLER	**$99**
87MACRO/DEBUG	**$199**
MICROSOFT FORTRAN V4	**$299**
RM FORTRAN	**$399**
LAHEY FORTRAN F77L	**$477**
MS or LATTICE C	**CALL**
STSC APL★PLUS/PC	**$450**
STSC STATGRAPHICS	**$675**
SPSS/PC+	**$695**
87SFL Scientific Functions	**$250**
87FFT	**$200**
OBJ→ASM	**$200**
PHOENIX PRODUCTS	**CALL**

287 Turbo™-10/12



287Turbo runs the 80287 at 10 or 12 MHz in the IBM PC AT, compatibles and the new Compaq 386 with 100% software compatibility.

10 MHz **$450**

12 MHz **$550**

PC Magazine **"Editor's Choice"**

8087 UPGRADES

All MicroWay 8087s include a one year warranty, complete MicroWay Test Program and installation instructions.

8087 5 MHz **$99**
For the IBM PC, XT and compatibles

8087-2 8 MHz **$154**
For Wang, AT&T, DeskPro, NEC, Leading Edge

80287-3 5 MHz **$159**

80287-6 6 MHz **$179**
For 8 MHz AT and compatibles

80287-8 8 MHz **$259**
For the 8 MHz 80286 accelerator cards

80287-10 10 MHz **$395**

80387-16 16 MHz **$495**

PC-PAL™ Programmer **$395**

64K 150ns .....**$15** **256K 150ns** **$36**

Call for great prices on V20 & V30

AT8™

Turns your AT into a high speed, multi-user Xenix business system!

8 port, intelligent serial controller with 3% response degradation. Includes 8 MHz 80186 with built in DMA **$1299**

MICROWAY SOFTWARE FOR LOTUS 1-2-3™

PowerDialer® Add-In for Lotus 1-2-3 Release 2. Automated telephone dialing from within 1-2-3. Adds least cost routing, automatic carrier selection and automated phone book worksheet. Builds customized dialing applications. Can be used with DesqView **$79**

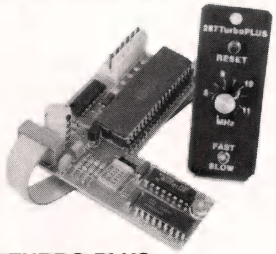
FASTBREAK™ employs the 8087 to increase the speed of Lotus 1-2-3™ Version 1A or 1A*. Users are reporting speed ups of between 3 and 36 to 1. When run with our NUMBER SMASHER accelerator card, recalculation speed ups of 10 to 30 are being reported **$79**

HOTLINK™ adds easy linking of spreadsheets to Lotus 1-2-3 Version 1A... **$99**

287 TURBO-PLUS™

Speeds up your AT

Adjustable 80286 Clock 6-12 MHz
10 MHz 80287 Clock
Plus Full Hardware Reset **$149**
Optional 80286-10 ..... **$175**



287 TURBO-PLUS
With 80287 10 MHz **$549**
With 80287 12 MHz **$629**

CALL (617) 746-7341 FOR OUR COMPLETE CATALOG

MicroWay
P.O. Box 79
Kingston, Mass.
02364 USA
(617) 746-7341

The World Leader in 8087 Support!

MicroWay Europe
32 High Street
Kingston-Upon-Thames
Surrey England KT1 1HL
Telephone: 01-541-5466

CIRCLE 300 ON READER SERVICE CARD

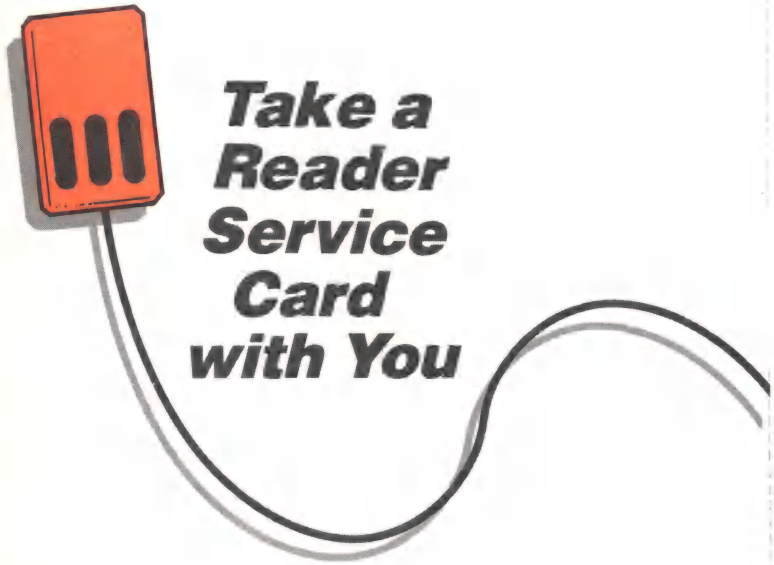
For Free Info ...

Start Here

Smart buyers start with *DDJ*'s free information card, a shopping center filled with information about the products and services advertised in this very issue: everything from software and systems to peripherals and professional support services.

And smart buyers can use this free information card to quickly and easily gather a comprehensive file of facts, figures and product specs to sort out competing claims. Using *DDJ*'s free information card can prevent you from making the wrong, costly buying decision.

Be a smart shopper. Complete and mail this postage paid card today!



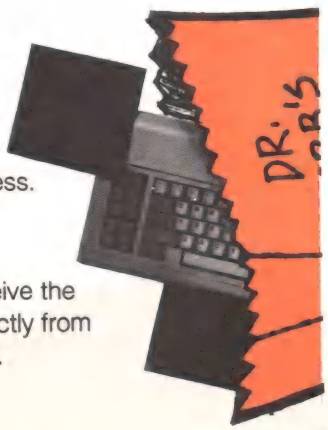
Take a Reader Service Card with You

It's Easy as ...

1. Circle the appropriate free information numbers, referring to the advertiser index for more information.

2. Fill in your name and address.

3. Mail today—postage is absolutely free. You'll receive the product information you need directly from the manufacturers, thanks to *DDJ*.



NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT #217, CLINTON, IOWA

POSTAGE WILL BE PAID BY ADDRESSEE

Dr. Dobb's Journal of Software Tools
FOR THE PROFESSIONAL PROGRAMMER

P.O. Box 2157
Clinton, Iowa 52735-2157

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT #217, CLINTON, IOWA

POSTAGE WILL BE PAID BY ADDRESSEE

Dr. Dobb's Journal of Software Tools
FOR THE PROFESSIONAL PROGRAMMER

P.O. Box 2157
Clinton, Iowa 52735-2157

Dr. Dobb's Journal of
Software Tools
FOR THE PROFESSIONAL PROGRAMMER

Use this card for FREE, FAST information about the products and services listed in this issue. Simply circle the appropriate numbers below.

1	26	51	76	101	126	151	176	201	226	251	276	301	326	351	376
2	27	52	77	102	127	152	177	202	227	252	277	302	327	352	377
3	28	53	78	103	128	153	178	203	228	253	278	303	328	353	378
4	29	54	79	104	129	154	179	204	229	254	279	304	329	354	379
5	30	55	80	105	130	155	180	205	230	255	280	305	330	355	380
6	31	56	81	106	131	156	181	206	231	256	281	306	331	356	381
7	32	57	82	107	132	157	182	207	232	257	282	307	332	357	382
8	33	58	83	108	133	158	183	208	233	258	283	308	333	358	383
9	34	59	84	109	134	159	184	209	234	259	284	309	334	359	384
10	35	60	85	110	135	160	185	210	235	260	285	310	335	360	385
11	36	61	86	111	136	161	186	211	236	261	286	311	336	361	386
12	37	62	87	112	137	162	187	212	237	262	287	312	337	362	387
13	38	63	88	113	138	163	188	213	238	263	288	313	338	363	388
14	39	64	89	114	139	164	189	214	239	264	289	314	339	364	389
15	40	65	90	115	140	165	190	215	240	265	290	315	340	365	390
16	41	66	91	116	141	166	191	216	241	266	291	316	341	366	391
17	42	67	92	117	142	167	192	217	242	267	292	317	342	367	392
18	43	68	93	118	143	168	193	218	243	268	293	318	343	368	393
19	44	69	94	119	144	169	194	219	244	269	294	319	344	369	394
20	45	70	95	120	145	170	195	220	245	270	295	320	345	370	395
21	46	71	96	121	146	171	196	221	246	271	296	321	346	371	396
22	47	72	97	122	147	172	197	222	247	272	297	322	347	372	397
23	48	73	98	123	148	173	198	223	248	273	298	323	348	373	398
24	49	74	99	124	149	174	199	224	249	274	299	324	349	374	399
25	50	75	100	125	150	175	200	225	250	275	300	325	350	375	400

September '87: Use before December 31, 1987

Name _____
Title _____
Company _____ Phone _____
Address _____
City/State/Zip _____

Dr. Dobb's Journal of
Software Tools
FOR THE PROFESSIONAL PROGRAMMER

Use this card for FREE, FAST information about the products and services listed in this issue. Simply circle the appropriate numbers below.

1	26	51	76	101	126	151	176	201	226	251	276	301	326	351	376
2	27	52	77	102	127	152	177	202	227	252	277	302	327	352	377
3	28	53	78	103	128	153	178	203	228	253	278	303	328	353	378
4	29	54	79	104	129	154	179	204	229	254	279	304	329	354	379
5	30	55	80	105	130	155	180	205	230	255	280	305	330	355	380
6	31	56	81	106	131	156	181	206	231	256	281	306	331	356	381
7	32	57	82	107	132	157	182	207	232	257	282	307	332	357	382
8	33	58	83	108	133	158	183	208	233	258	283	308	333	358	383
9	34	59	84	109	134	159	184	209	234	259	284	309	334	359	384
10	35	60	85	110	135	160	185	210	235	260	285	310	335	360	385
11	36	61	86	111	136	161	186	211	236	261	286	311	336	361	386
12	37	62	87	112	137	162	187	212	237	262	287	312	337	362	387
13	38	63	88	113	138	163	188	213	238	263	288	313	338	363	388
14	39	64	89	114	139	164	189	214	239	264	289	314	339	364	389
15	40	65	90	115	140	165	190	215	240	265	290	315	340	365	390
16	41	66	91	116	141	166	191	216	241	266	291	316	341	366	391
17	42	67	92	117	142	167	192	217	242	267	292	317	342	367	392
18	43	68	93	118	143	168	193	218	243	268	293	318	343	368	393
19	44	69	94	119	144	169	194	219	244	269	294	319	344	369	394
20	45	70	95	120	145	170	195	220	245	270	295	320	345	370	395
21	46	71	96	121	146	171	196	221	246	271	296	321	346	371	396
22	47	72	97	122	147	172	197	222	247	272	297	322	347	372	397
23	48	73	98	123	148	173	198	223	248	273	298	323	348	373	398
24	49	74	99	124	149	174	199	224	249	274	299	324	349	374	399
25	50	75	100	125	150	175	200	225	250	275	300	325	350	375	400

September '87: Use before December 31, 1987

Name _____
Title _____
Company _____ Phone _____
Address _____
City/State/Zip _____

For Free Info ...

Start Here

Smart buyers start with *DDJ*'s free information card, a shopping center filled with information about the products and services advertised in this very issue: everything from software and systems to peripherals and professional support services.

And smart buyers can use this free information card to quickly and easily gather a comprehensive file of facts, figures and product specs to sort out competing claims. Using *DDJ*'s free information card can prevent you from making the wrong, costly buying decision.

Be a smart shopper. Complete and mail this postage paid card today!

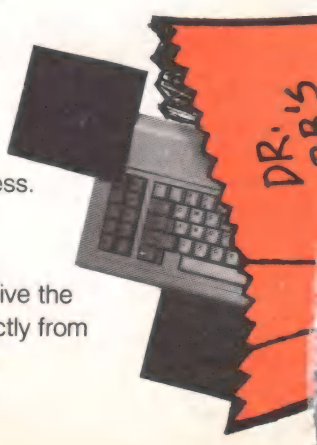
Take a Reader Service Card with You

It's Easy as ...

1. Circle the appropriate free information numbers, referring to the advertiser index for more information.

2. Fill in your name and address.

3. Mail today—postage is absolutely free. You'll receive the product information you need directly from the manufacturers, thanks to *DDJ*.



The Advertiser Index

Advertiser Name	Page No.	Circle No.
AI Architects	31	265
Aker Corporation	81	369
Aldebaran Laboratories	49	350
Alsys	46-47	273
Aptech Systems, Inc.	69	*
Austin Code Works	136	250
Blaise Computing	2	159
Borland International	1	161
Boston Software Works (The)	144	384
Bryte Computer	65	387
Burton Systems Software	85	212
C Tools	64	*
C Users Group	85	181
Cadre Technologies, Inc.	26	261
Cobalt Blue	73	370
Compu View	42	122
Creative Programming	61	*
Crosstalk Communications	C6	171
Custom Software Systems	109	268
DDJ Subscriptions	80	*
Datalight	9	203
Desktop A.I.	64	258
Digitalk	146	127
Ecosoft, Inc.	133	89
Enertec	87	346
Essential Software	C5	138
Fair-Com	131	93
Flexus	53	189
Galacticomm	98	362
Galacticomm	PB	342
GenSoft Systems	93	166
Gimpel Software	48	*
Greenleaf Software	111	97
Guidelines Software	33	351
Hauppauge Computer Works	147	274
Hi-Tech Software	148	376
Intel Corporation	4-5	179
JYACC	51	146
Logic Process Corporation	102	169
LogicPath	85	226
Logitech, Inc.	145	257
Lugaru Software Ltd.	132	135
M Street Software	66	276
MS-DOS Tools	64a	*
M&T Catalog of Books & Software Tools	PB	*
Machine Independent Software Corp.	103	294
Magma Systems	99	313
Manx Software Systems	7	108
Mark Williams Company	120	102
Meridian Software Systems	41	397
Metagraphics Software Corporation	83	392
MetaWare Incorporated	78	95
Micro Way	135	300
Microcompatibles	78	286
Microprocessors Unlimited	87	105
Microsoft	32a	380
Microsoft Press	115	125
Microsoft Press	114	126
MMC AD Systems	119	192
Mortice Kern Systems, Inc.	75	249
Nanosoft Associates	101	309
Nantucket Corporation	27	220
New BASICs	62	*
Norton Utilities (The)	134	243
Norton Utilities (The)	116 117	87
Oakland Group, Inc. (The)	110	227
Oasys	40	254
Oregon Software	63	357
Palo Alto Shipping	96	76
Periscope Co., Inc.	105	214
Pharlap	71	343
PIM Publications	79	136
Pioneering Controls Technologies, Inc.	53	191
PMI	89	239
Polytron Corporation	130	283
Prentice Hall	126	*
Programmer's Connection	149-151	129
Programmer's Connection	23	98
Programmer's Shop (The)	143	133
Quantum Computing	71	144
Quarterdeck Office Supplies	10	284
Raima Corporation	107	*
Rainbow Technologies	73	255
SAS Institute	141	*
Santa Rita Software	15	353
Scantel Systems Limited	96	391
Scientific Endeavors	66	210
Secom Information Products Co.	76	394
Seidl Computer Engineering	55	114
Semi-Disk Systems	52	85
Sharpe Systems Corporation	34	176
SLR Systems	77	78
Softcraft Inc.	11	113
Softfocus	79	259
Software Garden Inc.	59	314
Software Security, Inc.	35	170
Solution Systems	21	142
Solution Systems	112	152
Texas Instruments	16-17	*
Texas Instruments	129	*
Tool Makers (The)	73	319
True Basic	108	344
TSF (The Software Family)	97	230
Turbo Tech Report	67	119
Unipress Software	139	77
Vermont Creative Software	43	157
Wallsoft	91	90
Wendin	13	112
Whitewater Group (The)	50	282
Wordcraft	127	163
Zark Incorporated	77	164
ZyLAB	57	329

*This advertiser prefers to be contacted directly; see ad for phone number.

to. . . Oh my God—he did forget!).

Stan Kelly-Bootle
25 Parkwood
Mill Valley, CA 94941

Dear *DDJ*,

Brian Anderson states that "often the only control structures available in assembly language are the conditional and unconditional jump and the call to subroutine." The 68000 and most other modern processors have looping structures, generally provided for use by high-level language compilers. Therefore, I offer my Example 2, below, as a replacement for his. Also, notice that this code does not get stuck in an endless loop, unlike its predecessor.

I would like to thank Mr. Anderson for his interesting and otherwise excellent article.

Matthew Siegel
192 Belvedere St., #9
San Rafael, CA 94901-4748

Dear *DDJ*,

I guess I still have a bit of egg on my face—it seems that I can't program my way out of an infinite loop (at least when it comes to 68000 assembly language).

The 68000 example that I cited in my Viewpoint won't work because I forgot to increment the index variable inside the loop. My wife thought that, because this mistake supports my point (sort of), I should claim that I planted the error just to see if anyone would catch it.

Please let me assure you, the mistake was an honest one. My apologies to *DDJ* readers (especially 68000 hackers).

Brian Anderson
5105 Lorraine Ave.
Burnaby, BC
Canada V5G 2S3

Math and Programming

Dear *DDJ*,

I feel I must take issue with Allen Holub concerning his Viewpoint in the April 1987 issue of *DDJ*.

To begin with, from personal experience, I can draw a strong correlation between advanced mathematics and the art of programming. There is a definite parallel between juggling a large system of algebraic equations in your head and trying to maintain the purpose and use of many interacting variables in a computer program. Obviously implementations of some of the purer algorithms in computer science such as queueing theory, graphs and trees, sorting algorithms, vector math, automata theory, frequency analysis, formal logic, cubic splines, compression algorithms, minimax and Bayes decision theory, branch and bound problems, probabilistic and statistical concepts, game theory, and all aspects of operations research require a deep understanding of algebraic notation, linear algebra, and many other forms of abstract representation.

Allen implies that mathematics is nothing more than applying a set of memorized rules to a problem. I beg to differ if he feels that solving a third-order differential equation or a partial derivative is not a creative problem solving process. Those "memorized rules" are less rules and more approaches and guidelines with which to tackle the problem as stated. I find solving calculus and differential equations not at all unlike trying to come up with my own algorithms for a computer simulation problem. I feel that those students who could not tackle and solve advanced mathematics would also have great difficulty in implementing their own fresh approaches to new and yet unsolved computer science problems.

In agreement with Allen, I would state that if a student's goal is to become an applications programmer who is never responsible for an original algorithm but who instead simply implements code and algorithms found in books, then by all means forego anything beyond Algebra I. Allen covers himself by stating that math is not a prerequisite for programming as it is for engineering. What does he think the name computer scientist means anyway? And what kind of jobs do you think computer scientists have? They work on developing new approaches in medicine, vision processing, knowledge

```
*************************************************************
*
* 68000 assembly language FOR loop
*
* INPUT
* The first element to clear is FIRST, the last element LAST.
* Character data begins at DATA.
*
* REGISTER USAGE
* A0 points to the current element of the array.
* D0 = one less than the number of elements left to clear.
*
*************************************************************
*
FIRST  DS.W 1
LAST   DS.W 1
DATA   DS.B 500
*
*
CLEAR  LEA   DATA,A0        ; A0 points to character data
       MOVE.W FIRST,D0      ; D0 = first element to clear
       LEA   0(A0,D0.W),A0  ; A0 points to first element to clear
       MOVE.W LAST,D0       ; D0 = number of elements to clear
       SUB.W  FIRST,D0      ;   (less 1, for DBcc loop use)
LOOP   CLR.B  (A0)+         ; clear an element and advance
       DBF    D0,LOOP       ; repeat
*
       END
```

Example 2: *Loop structure correction for Brian Anderson's 68000 assembly code*

representation, flight simulation, computer games, the defense industry, and all forms of real-world simulation and control. One hardly needs a computer scientist to write a dBASE III application, yet the fields mentioned above seek out and demand a computer scientist with heavy mathematics background. This group of programmers does not make up a small specialized percentage but instead represents what a computer science degree is all about. Would you hire an MIT computer science graduate who had no more than high-school geometry to his or her credit? Neither would I.

If Allen rewrote the entire Viewpoint and replaced each occurence of "computer scientist" with "data processing programmer," it would be a valid and important commentary. A dBASE III, 4GL, or COBOL programmer has little need for calculus, but for those of us breaking new frontiers in image processing, problem solving, and other areas of computer science, the need for a strong background in mathematics and formal symbolic representation is clear.

John W. Ratcliff
2510 LaCaracas
St. Louis, MO 63114

Allen Holub replies:

There are several ways to learn how to manage large systems, and I still believe that mathematics is among the poorest of these, primarily because of the amount of background that you need just to get started. It takes a year and a half of college-level math to get to the point where you can start solving third-order differential equations, but most people (hopefully) know the rudiments of English composition before they get out of high school. Moreover, most holders of CS bachelors degrees don't know how to solve differential equations at all, for the simple reason that the courses aren't usually required. Mr. Ratcliff is correct in saying that higher mathematics can be useful to a programmer. Mathematics at the undergraduate level is not. It seems to me a waste of time to acquire the skills necessary to understand differential equations if all you really need to do is understand how to approach complex systems. It's not that mathematics won't get you there eventually but that there are better and faster ways to acquire the same skills, notably English composition.

Mr. Ratcliff's also contends that a strong background in "formal symbolic representation" is necessary. Again, I disagree. An ability to work with abstraction is, of course, necessary, but remember that logic started out life as a branch of philosophy, not of mathematics. Much of the basic work in compiler theory was done by linguists (such as Noam Chomsky at MIT), not by mathematicians. More often than not, a "formal symbolic representation" serves to obfuscate, rather than clarify. Good examples of this obfuscation can be found in the "dragon" book (Aho, Sethi, and Ulman), written by mathematicians and used in most compiler-design classes (even, I'm reluctant to admit, in the one that I teach). Aho spends pages inventing "formal symbolic representations" and then spends five lines actually explaining something useful. I'd rather spend half an hour reading a clear description of a process in English than spend the same half hour trying to decipher five lines of formal symbols. More to the point, I've found that students who find Aho incomprehensible have no trouble at all understanding the concepts, once these concepts are presented to them in a clear way that doesn't use Aho's formal symbolic notation.

Call for Recipes

Dear *DDJ*,

In his boffo review of our book, *Numerical Recipes: The Art of Scientific Computing* (May 1987), Joe Marasco mentions that we want to hear from readers who would like to see a C version of the book and source-code "recipes."

Actually, preparation of *Numerical Recipes in C* is well underway. In fact, we would like to hear from *DDJ* readers with an interest in beta-testing the C recipes. We'll happily send free beta diskettes to the first hundred people who respond, plus an additional number to readers who can describe (in a sentence or two) their strong qualifications.

We view *Numerical Recipes* as a cooperative project between authors and users. It's time to get good, cheap, open-source numerical software into the C world, before the vendors of proprietary object-only libraries get too established, as was historically the case in the FORTRAN world (much to the grief of FORTRAN programmers).

William Press
Numerical Recipes Software
P.O. Box 243
Cambridge, MA 02238

Correction

Dear *DDJ*,

Thank you for covering PC-MOS/386 from The Software Link in two articles in your July 1987 issue. Unfortunately, some of the information in each article is incorrect. I'd like to take this opportunity to bring it to your attention.

In the article "Developing 80386 Applications . . . Today," the prices given for PC-MOS/386 are wrong. The single-user/multitasking version is $195, the five-user version is $595, and the twenty-five-user version is $995.

In the 16-Bit Software Toolbox column The Software Link is erroneously referred to on page 106 as "a company previously known for its copy-protection schemes." The Software Link has never been involved in any type of copy-protection—we develop and manufacture multiuser/multitasking software.

Thank you for the opportunity to point out these errors of fact. Again, we appreciate your editorial coverage. Our programming staff considers *Dr. Dobb's* a valuable resource and we are grateful for your coverage of our products.

Colleen G. Goidel
The Software Link
3577 Parkway Ln.
Atlanta, GA 30092

DDJ

SAS Institute Inc. Announces

Lattice C Compilers for Your IBM Mainframe

Two years ago...

SAS Institute launched an effort to develop a subset of the SAS® Software System for the IBM Personal Computer. After careful study, we agreed that C was the programming language of choice. And that the Lattice® C compiler offered the quality, speed, and efficiency we needed.

One year ago...

Development had progressed so well that we expanded our efforts to include the entire SAS System on a PC, written in C. And to insure that the language, syntax, and commands would be identical across all operating systems, we decided that all future versions of the SAS System—regardless of hardware—would be derived from the same source code written in C. That meant that we needed a C compiler for IBM 370 mainframes. And it had to be good, since all our software products would depend on it.

So we approached Lattice, Inc. and asked if we could implement a version of the Lattice C compiler for IBM mainframes. With Lattice, Inc.'s agreement, development began and progressed rapidly.

Today...

Our efforts are complete—we have a first-rate IBM 370 C compiler. And we are pleased to offer this development tool to you. Now you can write in a single language that is source code compatible with your IBM mainframe and your IBM PC. We have faithfully implemented not only the language, but also the supporting library and environment.

Features of the Lattice C compiler for the 370 include:

- **Generation of reentrant object code.** Reentrancy allows many users to share the same code. Reentrancy is not an easy feature to achieve on the 370, especially if you use non-constant external variables, but we did it.
- **Optimization of the generated code.** We know the 370 instruction set and the various 370 operating environments. We have over 100 staff years of assembler language systems experience on our development team.
- **Generated code executable in both 24-bit and 31-bit addressing modes.** You can run compiled programs above the 16 megabyte line in MVS/XA.
- **Generated code identical for OS and CMS operating systems.** You can move modules between MVS and CMS without even recompiling.
- **Complete libraries.** We have implemented all the library routines described by Kernighan and Ritchie (the informal C standard), and all the library routines supported by Lattice (except operating system dependent routines), plus extensions for dealing with 370 operating environments directly. Especially significant is our byte-addressable Unix®-style I/O access method.
- **Built-in functions.** Many of the traditional string handling functions are available as built-in functions, generating in-line machine code rather than function calls. Your call to move a string can result in just one MVC instruction rather than a function call and a loop.

In addition to mainframe software development, you can also use our new cross-compiler to develop PC software on your IBM mainframe. With our cross-compiler, you can compile Lattice C programs on your mainframe and generate object code ready to download to your PC.

With the cross-compiler, we also offer PLINK86™ and PLIB86™ by Phoenix Software Associates Ltd. The Phoenix link-editor and library management facility can bind several compiled programs on the mainframe and download immediately executable modules to your PC.

Tomorrow...

We believe that the C language offers the SAS System the path to true portability and maintainability. And we believe that other companies will make similar strategic decisions about C. Already, C is taught in most college computer science curriculums, and is replacing older languages in many. And almost every computer introduced to the market now has a C compiler.

C, the language of choice...

C supports structured programming with superior control features for conditionals, iteration, and case selection. C is good for data structures, with its elegant implementation of structures and pointers. C is conducive to portable coding. It is simple to adjust for the size differences of data elements on different machines.

Continuous support...

At SAS Institute, we support all our products. You license them annually; we support them continuously. You get updates at no additional charge. We have a continuing commitment to make our compiler better and better. We have the ultimate incentive—all our software products depend on it.

For more information...

Complete and mail the coupon today. Because we've got the development tool for your tomorrow.

SAS®

SAS Institute Inc.
SAS Circle, Box 8000
Cary, NC 27511-8000
Telephone (919) 467-8000 x 7000

I want to learn more about:

- ☐ the C compiler for MVS software developers
- ☐ the C compiler for CMS software developers
- ☐ the cross-compiler with PLINK86 and PLIB86

today...so I'll be ready for tomorrow.

Please complete or attach your business card.

Name _____

Title _____

Company _____

Address _____

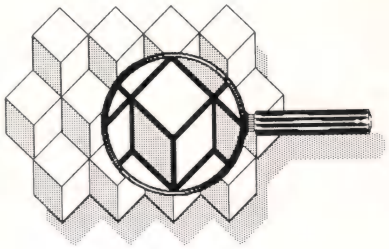
City _____ State _____ ZIP _____

Telephone _____

Mail to: SAS Institute Inc., Attn: CC, SAS Circle, Box 8000, Cary, NC, USA. 27511-8000. Telephone (919) 467-8000, x 7000

DDJ 9/87

OF INTEREST



PS/2 Add Ons

Alloy Computer Products has announced three products for the PS/2 line: an internal tape drive and two adapters to connect its external tape drives and other products to the PS/2 machines. Reader Service No. 17.
Alloy Computer Products Inc.
100 Pennsylvania Ave.
Framingham, MA 01701
(617) 875-6100

Rodime has announced a hard disk on a card for the PS/2 Model 30, which it claims is "the only way for Model 30 users to get more than 20 megabytes of internal storage." The suggested retail price is $1,495. Reader Service No. 18.
Rodime Inc.
Peripheral Systems Division
29525 Chagrin Blvd., Ste. 214
Pepper Pike, OH 44122
(216) 765-8414

CMS Enhancements has exhibited external hard-disk subsystems for the PS/2 Model 30 as well as for the entire PS/2 line and for the Macintosh SE. CMS also has the first 5¼-inch floppy drive for the PS/2, which should help all those early adopters move their software over to the PS/2 hardware. Reader Service No. 19.
CMS Enhancements Inc.
1372 Valencia Ave.
Tustin, CA 92680
(714) 259-9555

Kodak's diskette subsidiary, **Verbatim**, has announced a 2-megabyte 3.5-inch diskette (formatted capacity 1.44 megabytes). Reader Service No. 20.
Verbatim Corp.
1200 W.T. Harris Blvd.
Charlotte, NC 28213
(704) 547-6500

Development Software

Sterling Castle, a new publisher of PC software, has introduced the Blackstar C function library designed to support the new ANSI standard and Microsoft, Version 3.0/4.0, and Lattice 3.0 C Compilers. Reader Service No. 21.
Sterling Castle Software
702 Washington St., Ste. 174
Marina del Rey, CA 90292
(213) 206-3020

The macro processing program SmartKey is evolving closer to a programming language as of its new version, 5.2, which adds context sensitivity and conditional processing. Programmer **Nick Hammond** wrote SmartKey back in 1979, and it is the original macro processor. It costs $69.95. Reader Service No. 22.
Software Research Technologies Inc.
2130 South Vermont Ave.
Los Angeles, CA 90007
(213) 737-7663

Alan Weiner has taken macro generation a step further. He has developed a memory-resident programming language, which he calls the Weiner Shell and which generates memory-resident programs. The program is written in assembly language, takes up less than 50K, and supports the LIM expanded memory spec, so any program written with it has access to up to 8 megabytes of LIM memory as well. It supports DOS interrupts, user I/O, arrays, and floating-point math. The idea of writing memory-resident programs on the fly that play around with DOS interrupts is scary, but Alan claims the product itself is robust. Its price is $199. Reader Service No. 23.
Gryphon Microproducts
P.O. Box 6543
Silver Spring, MD 20906
(301) 384-6868

Caro Research has a C code generator, developed by an outfit called Chancelogic PLC, called Pro-C that supports various ISAM file handlers; produces stand-alone C programs rapidly; and requires no end-user run-time system, royalties, or license. Reader Service No. 24.
Caro Research Associates
202 South 22nd St.
Tampa, FL 33605
(813) 248-0852

Books and Stuff

We don't know why the company is called **Rabbit**, but we are compelled by this nomenclature to tell you that Rabbit has announced publication of its *Portable C and Unix Programming*, a 240-page reference guide for programmers writing applications in C or in the Unix environment. The book is available from Prentice-Hall for $21.95. Reader Service No. 25.
Rabbit Software Corp.
Great Valley Corporate Center
Seven Great Valley Parkway East
Malvern, PA 19355
(215) 647-0440

AT&T also wants to educate you. It has announced a series of videotapes on Unix System V, Release 3. The tapes of possible interest to *DDJ* readers cover C and command shell programming and can be leased for $300–375 a month or purchased for more. Reader Service No. 26.
AT&T
Videotape Library
(800) 247-1212

Windows Applications

Palantir Software has a shelf full of Windows applications, including word processing, spelling checking, scheduling, report generation, spreadsheeting, drawing, graphics scanning, and communications. Reader Service No. 27.
Palantir Software
12777 Jones Rd., Ste. 100
Houston, TX 77070
(713) 955-8880

Among the announced Windows applications at Comdex were two from **hDC**: an applications organizer called ClickStart, and EGA-16, a driver that doubles the number of displayable colors under Windows. The idea behind ClickStart seems to be that corporate users of Windows will need password protection of applications, turnkey menu selections, and restricted access to DOS functions. The vision of a traditional DP/MIS user interface on top of a point-and-shoot graphical user interface on top of the

venerable A> prompt seemed a bit much, but later the same day we found another vendor doing likewise. EGA-16 is $24.95, and ClickStart is $79.95. Reader Service No. 28.
hDC Computer Corp.
8405 165th Ave. NE
Redmond, WA 98052
(212) 475-5550

Modems

Prices of 2,400-bps modems are coming down a bit. **Okidata** has announced two new 2,400-bps modems at $599 (external) and $549 (internal), with automatic adaptive equalization to ameliorate the problems of line noise. Reader Service No. 29.
Okidata
532 Fellowship Rd.
Mount Laurel, NJ 08054
(609) 235-2600

The **Zoom**/Modem PC 2400 HC is an internal 300/1,200/2,400-bps IBM PC, IBM PC/XT, IBM PC/AT, and compatible modem with a suggested retail price of $199. It supports Bell 103a, 212a, and CCITT v.22 bis protocols and uses the Hayes AT command set. Reader Service No. 30.
Zoom Telephonics Inc.
207 South St.
Boston, MA 02111
(617) 423-1072

Displays

Jeff Duntemann, editor of Borland's new *Turbo Techniques*, often nags software vendors about support for full-screen displays. Jeff owns a Genius 15-inch full-page display and has the radical idea that software developers should not penalize him for owning a good monitor. **Micro Display Systems** announced at Comdex a line of 19-inch Genius full-screen displays using the TI 34010 graphics coprocessor. Reader Service No. 31.
Micro Display Systems Inc.
1310 Vermilion St.
P.O. Box 455
Hastings, MN 55033
(612) 437-2233

Cornerstone Technology has introduced the Vista 1600, a $2,195 19-inch monitor for the Macintosh II that displays 1,600 × 1,280-resolution, the Mac's max. Its noninterlaced screen has a 200-MHz bandwidth and refreshes at 67 Hz. The monitor comes with a NuBUS controller card that occupies one slot in the Mac II.

Cornerstone has also announced a version of the monitor for PCs and 386 machines for $2,395. Reader Service No. 32.
Cornerstone Technology
175A East Tasman Dr.
San Jose, CA 95134-1620
(408) 433-1600

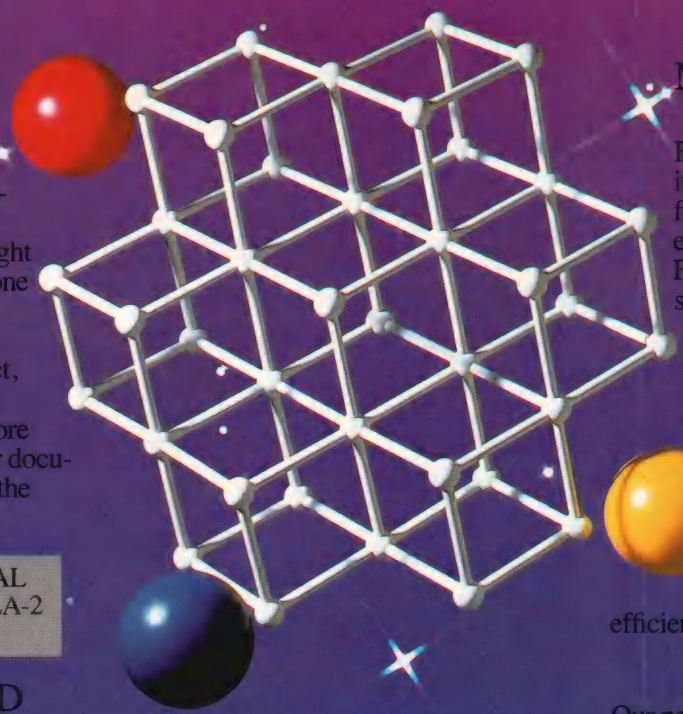
Hitachi has announced more hi-res monitors, including a 20-inch 1,280 × 1,024-model with RGB input and various scanning frequencies for $3,995 suggested retail. Reader Service No. 33.
Hitachi Sales Corp. of America
401 West Artesia Blvd.
Compton, CA 90220
(213) 537-8383

Amdek has announced a family of color and monochrome displays compatible with the PS/2's VGA spec. Models 432 (monochrome) and 732

SOFTWARE ENGINEERING COMES OF AGE.

ANNOUNCING LOGITECH MODULA-2 VERSION 3.0

Modula-2 is the language of choice for modern software engineering, and LOGITECH Modula-2 is the most powerful implementation available for the PC. The right language and the right tools have come together in one superior product. Whether you're working on a small program or a complex project, with LOGITECH Modula-2 Version 3.0 you can write more reliable, maintainable, better documented code in a fraction of the time at a fraction of the cost.



FREE TURBO PASCAL TO LOGITECH MODULA-2 TRANSLATOR

NEW, IMPROVED DEBUGGERS

Time gained with a fast compiler can be lost at debug time without the right debugging tools. With the powerful Logitech Modula-2 Debuggers you can debug your code *fast*, and dramatically improve your overall project throughput. The Post Mortem Debugger analyzes the status of a program after it has terminated while the dynamic, Run Time Debugger monitors the execution of a program with user-defined breakpoints. With their new, mouse based, multiple-window user interface these powerful debugging tools are a pleasure to use.

NEW, INTELLIGENT LINKER

Links only those routines from a particular module that you need, so you eliminate unreferenced routines and produce smaller, more compact executable files.

NEW, IMPROVED COMPILER

Faster and more flexible. Now its DOS linker compatible object files (.OBJ) can be linked with existing libraries in C, PASCAL, FORTRAN and ASSEMBLER — so you can build on previous development and put the power of LOGITECH Modula-2 to work for you right now. Fully supports Wirth's latest language definition, including LONGINT and LONGSET, which provides large set support including SET of CHAR. Provides optimization for tighter, more efficient code generation.

NEW EDITOR

Our new, mouse based editor is fully integrated, easy to learn, fast and easy to use, and very customizable. Its multiple, overlapping windows and color support make it easy to manage parts of one file or several files on the screen at one time. You'll love using it — with or without a mouse.

Call for information about our VAX/VMS version, Site License, University Discounts, Dealer & Distributor pricing.

To place an order call toll-free:

800-231-7717

In California:

800-552-8885

☐ LOGITECH Modula-2 V. 3.0 Compiler Pack — **$99**
Compiler in overlay and fully linked form, Linkable Library, Post Mortem Debugger, Point Editor

☐ LOGITECH Modula-2 V. 3.0 Toolkit — **$169**
Library sources, Linker, Run Time Debugger, MAKE, Decoder, Version, XRef, Formatter

☐ LOGITECH Modula-2 V. 3.0 Development System — **$249**
Compiler Pack plus Toolkit

☐ Turbo Pascal to Modula-2 Translator — **FREE**
With Compiler Pack or Development System

☐ Window Package — **$49**
Build true windowing into your Modula-2 code.

☐ Upgrade Package
Call LOGITECH for information or to receive an order form.

Add $6.50 for shipping and handling. California residents add applicable sales tax. Prices valid in U.S. only. Total Enclosed $_____

☐ VISA ☐ MasterCard ☐ Check Enclosed

Card Number _____ Expiration Date _____

Signature _____

Name _____

Address _____

City _____ State _____

Zip _____ Phone _____

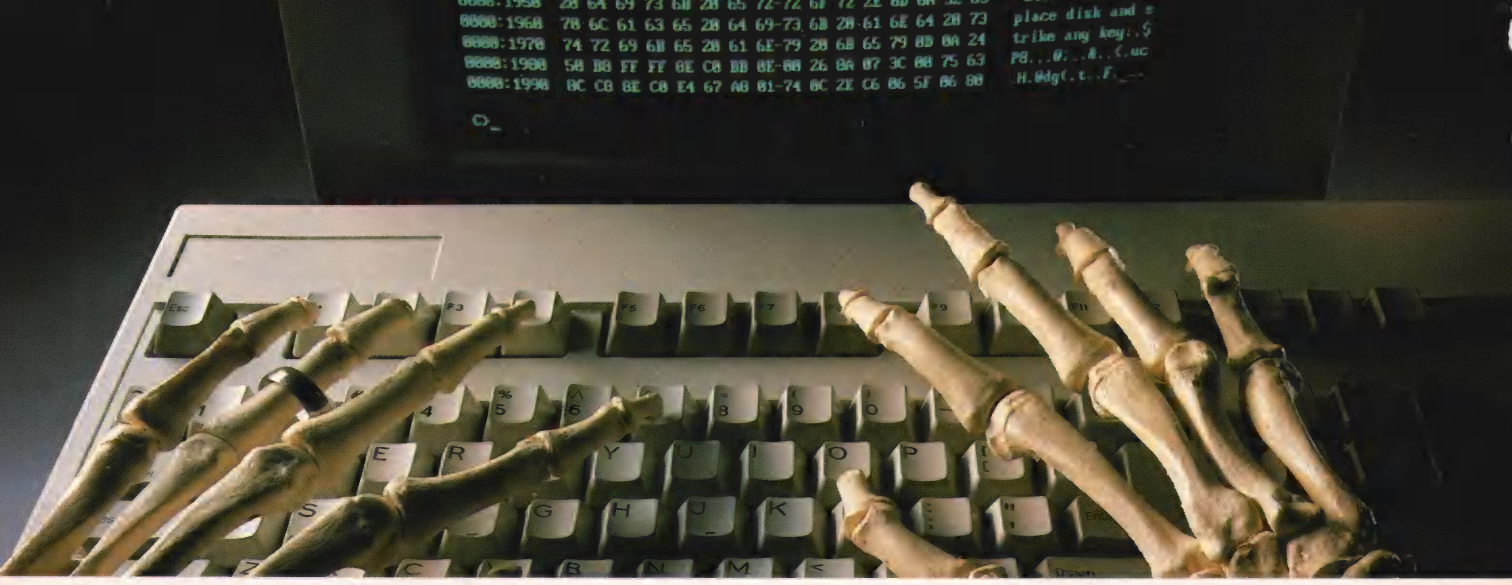
LOGITECH

LOGITECH, Inc.
6505 Kaiser Drive, Fremont, CA 94555
Tel: 415-795-8500

In Europe: LOGITECH, Switzerland
Tel: 41-21-87-9656 Telex 458 217 Tech Ch

In the United Kingdom: LOGITECH, U.K.
Tel: 44908-368071 Fax: 44908-71751

CIRCLE 257 ON READER SERVICE CARD



USING THE WRONG LANGUAGE CAN BE MURDER. SPEAK SMALLTALK/V.

Let's talk languages. Programming languages like Turbo Pascal, C or Basic can be killers. To many, they're foreign, complex, and generally intimidating. Mistakes can be deadly.

With Smalltalk/V, you have an elegantly simple solution that puts the power and majesty of a major AI programming language on your PC or compatible. It makes no difference if you're an experienced programmer or just getting started. Smalltalk/V gives you an easy-to-use and flexible programming tool.

This is the same language used by leading software companies for their new product development. There are sound reasons for this. Smalltalk/V offers a totally integrated programming environment using the premier object-oriented language. You use natural language rather than complex programming codes. It puts Macintosh-type graphic features on a PC including overlapping windows, bit-mapping, pop-up menus, and a mouse interface. More than mere window dressing, Smalltalk/V delivers fully interactive windows that are easy to build and quick to modify.

But don't just take our word on it. Hear what the experts have to say:

"This is the real thing folks. A super Smalltalk like this turns your PC into a hot workstation. It's fantastic... Highly recommended."

John Dvorak
Contributing Editor
PC Magazine

"The tutorial provides the best introduction to Smalltalk available."

Dr. Andrew Bernat
AI Expert Magazine

"Smalltalk/V is the highest performance object-oriented programming system available for PCs."

Dr. Piero Scaruffi
Chief Scientist
Olivetti Artificial
Intelligence Center

Today, thousands of professionals, scientists and engineers are using Smalltalk/V to solve both simple and expert problems. Giving them a new dimension in computer applications for their PC.

Put new life into your PC by calling toll free 1-800-922-8255 and ordering Smalltalk/V today. Smalltalk/V by Digitalk, Inc., 9841 Airport Blvd., Los Angeles, CA 90045. (213) 645-1082.

$99.95

Smalltalk/V comes with 10 starter applications including Prolog and each Application Pack adds several more. All source code is included. Supports 640 x 480 color graphics with color extension pack.

Smalltalk/V requires DOS and 512K RAM on IBM PC/AT/PS or compatibles and a CGA, EGA, Toshiba T3100, Hercules, or AT&T 6300 graphic controller. A Microsoft or compatible mouse is recommended. Not copy protected.

TO ORDER CALL 1-800-922-8255 TODAY.

60-DAY MONEY-BACK GUARANTEE*

Send check, money order, or credit card information to: Digitalk, Inc., 9841 Airport Boulevard, Los Angeles, CA 90045.

Credit Card ☐ VISA ☐ Mastercard

Card number: _____

Expiration date: _____

Name: _____

Street Address: _____

City/State/Zip: _____

Smalltalk/V	$99.95
RS-232 Communications Application Pack	$49.95
EGA/VGA Color Extension Pack	$49.95
"Goodies" Application Pack	$49.95
SPECIAL OFFER: Smalltalk/V and all 3 packs only	$199.95
Shipping and handling (Outside North America	$5.00 ($15.00)
California residents add applicable sales tax	
TOTAL	$_____

*Unconditional 60-day money-back guarantee. Simply return to Digitalk, Inc. and your refund will be immediately forwarded to you.

Smalltalk/V

digitalk inc.

CIRCLE 127 ON READER SERVICE CARD

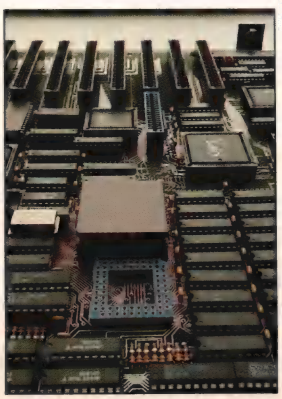
SAVE YOUR PC FROM EARLY RETIREMENT.

GET HAUPPAUGE'S NEW 386 MOTHERBOARD.

386 SPEED–ONLY $1,495

Give your PC a new lease on life! With our industry first 386 MotherBoard, your PC, PC/XT or compatible will revel in speeds equal to the Compaq DeskPRO 386. And faster. Because we've built in 1 Megabyte of high speed RAM and a 387 math coprocessor socket for speeds that will knock you off your rocker.

To keep retirement at bay, our 386 MotherBoard is compatible with the PC/AT (BIOS and I/O) — allowing you to run the new generation of DOS, OS/2. We've also included a 16-bit expansion slot that accommodates the latest I/O expansion card. No accelerator card can give you so much versatility.



With 386 power and true AT software compatibility, your business, desktop publishing and engineering applications will get a boost to boast about! **Technical Features** ▪ 16 MHz 80386 ▪ 1 Megabyte of 100 nsec 4-way interleaved RAM ▪ PC/AT compatible I/O and BIOS for support of OS/2 ▪ Seven 8-bit expansion slots ▪ Two 16-bit expansion slots ▪ One 32-bit RAM expansion slot ▪ Optional 16 MHz 80387 math coprocessor ($695)

Put the power of the 386 into your IBM PC for 1/4 the cost of a 386 computer. And put off your PC's retirement. For more information on our easy-to-install Motherboard, call **1 (800) 443-6284.** In New York, call **(516) 360-3827.**

Hauppauge Computer Works, Inc.
358 Veterans Memorial Highway,
Commack, New York 11725

Hauppauge!

Trademarks: IBM PC AT and PC-XT, IBM, DeskPro 386, Compaq Computers

CIRCLE 274 ON READER SERVICE CARD

(color) display 640 × 480 (VGA), 640 × 400 (CGA), or 640 × 350 (EGA) pixels and sell for a suggested retail price of $245 and $625, respectively. Reader Service No. 34.
Amdek Corp.
1901 Zanker Rd.
San Jose, CA 95112
(408) 436-8570

Tseng Labs has announced a new VLSI chip for IBM VGA graphics, for which the company expects to show a prototype this month. The ET3000 chip will support all VGA graphics as well as MGA, CGA, and EGA. Reader Service No. 35.
Tseng Laboratories Inc.
10 Pheasant Run
Newtown, PA 18940
(215) 968-0502

CAD/CAM/CAE

Vector Automation claims that its CADMAX Version 3.0 software for 386 workstations, formerly available on MicroVAX II, is the first CAD software to take full advantage of the 386. The price is $3,350 for a 2-D version. Reader Service No. 36.
Vector Automation Inc.
Village of Cross Keys
Baltimore, MD 21210
(301) 433-4200

Modgraph's Pegasus is a graphics card that supports EGA (or CGA, MGA, or Hercules) and high-resolution graphics on one monitor of an IBM PC, IBM PC/XT, IBM PC/AT, or compatible computer, for CADD work and the like. The hi-res support includes an 82786 graphics engine, there is a chip set supporting EGA, and the board discerns which mode you want. Reader Service No. 37.
Modgraph
149 Middlesex Turnpike
Burlington, MA 01803
(617) 229-4800

The Great Western Software Company has a companion software product to AutoCAD called AutoBoard System II for use in the production of printed circuit boards. This is TGWSC's third automatic routing package for microcomputers; it's been at it for a while. Reader Service No. 38.
The Great Western Software Co.
207 West Hickory St., Ste. 202
Denton, TX 76201
(817) 383-4434

The **MSA Group**, developer of a 2-D CAD system for PCs, took the name of its product, TurboCAD, seriously, and, following the Borland lead, cut the product's price from $395 to $99 for all modules. Reader Service No. 39.
MSA Group
12021 Wilshire Blvd., Ste. 370
West Los Angeles, CA 90025
(213) 473-8711

Computervision has announced Version 3.0 of its PC-based 3-D CAD/CAM package, Personal Designer. New features include multiple views, improved dimensioning capability, and an undo feature. Reader Service No. 40.
Computervision Corp.
100 Crosby Dr.
Bedford, MA 01730
(617) 275-1800

DDJ

HARD FACTS:

We're Programmer's Connection, your best one-stop source for quality programmer's development tools for IBM personal computers and compatibles. Here are some important facts you should know about us and other dealers in our industry.

FACT:
FREE Shipping. Shipping to U.S. customers is FREE via UPS Ground. If you want your order shipped via an express service, we'll only charge you the shipping carrier's standard rate with no special fees. Some dealers charge extra for shipping and then add rush charges for shipping via express services. Others may advertise "free" shipping but make up for it by charging extra handling fees.

FACT:
Credit Cards. We'll charge your credit card only when we actually ship your order. Some dealers would charge your credit card at the time you place your order. This could leave you waiting for your shipment for weeks or months while they use your money interest-free.

FACT:
Discounts. We discount all software products — even special order items. Every product in our advertised price list is shown with its list price and discounted price. We want you to know exactly how much you'll save on every product. We don't try to fool you by discounting some products and charging full retail for others.

FACT:
Consistent Prices. We extend the same current prices to every customer regardless of where they see our ad. Some dealers vary prices in different ads and then ask you to mention which one you saw. This technique allows them to charge you the highest prices possible.

FACT:
No Hidden Charges. The discount prices you see on the next two pages are all you pay. We don't charge extra for UPS Ground shipping, credit cards, COD orders, purchase orders, sales tax (except Ohio) or special handling (except for non-Canadian international orders).

FACT:
Guarantees. We offer FREE 30-day no-risk return guarantees and 30-day evaluation periods on most of our products. Some dealers have no return options while others often charge restocking fees of 15% or more.

FACT:
Quality Products. Our product line consists of hundreds of high quality software development tools specifically for IBM Personal Computers and compatibles. While some dealers try to carry every software product ever written, we carry only those that meet our very high standards for quality and value.

FACT:
Latest Versions. The products we carry are the latest versions and come with the same manufacturer's technical support as if buying direct. While some dealers may participate in the software gray market, we're authorized to sell every product we carry.

FACT:
Large Inventory. We have one of the largest inventories of programmer's development products in the industry. Most orders are shipped within 24 hours. And if we don't have a product in stock, we'll get it for you fast.

FACT:
Meticulous Packaging. We'll give your shipment the extra protection needed to reach you in the best possible condition. First we'll protect your products from moisture by wrapping them in plastic. Then we'll insulate your box with high quality bubble-wrapping instead of the messy styrofoam chips that many other dealers use. Finally, we'll double-tape your box for extra strength.

FACT:
Independence. Since we're not directly affiliated with any software publisher or developer, we can give you sound, unbiased advice. Unlike some dealers who have a special interest in promoting only certain products, we'll give you an objective look at the products we carry.

FACT:
Noncommissioned Staff. Our courteous sales staff is always ready to help you. And if you aren't sure about your needs, our knowledgeable technical staff can give you sound, objective advice. Because they are noncommissioned, you won't be pressured into making a purchase.

As you can see, we're different from the other dealers in our industry. Our customers keep coming back because we consistently provide the highest quality service and the lowest prices. So call us today and experience the differences for yourself.

PROGRAMMER'S CONNECTION
QUALITY INTEGRITY Service SUPPORT PRICE

Turn the page for our product list and ordering information.

CIRCLE 129 ON READER SERVICE CARD

ai - expert systems

	List	Ours
1st-CLASS *by Programs in Motion*	495	399
EXSYS *Development Software by EXSYS*	395	309
EXSYS Runtime System	600	469
Logic-Line Series *All varieties by Thunderstone*	CALL	CALL

ai - lisp language

GCLISP *Golden Common LISP by Gold Hill*	495	CALL
GCLISP 286 Developer *by Gold Hill*	1190	CALL
Microsoft LISP *Common LISP*	250	149
QNIAL *Combines LISP & APL by NIAL Systems*	CALL	CALL
TransLISP PLUS *from Solution Systems*	195	125

ai - prolog language

Arity Combination Package	1095	979
Expert System Development Pkg	295	229
File Interchange Toolkit	50	44
PROLOG Compiler & Interpreter	650	569
Screen Design Toolkit	50	44
SQL Development Package	295	229
Arity PROLOG Interpreter	295	229
Arity Standard Prolog	95	77
LPA microPROLOG *All Varieties*	CALL	CALL
MPROLOG Language Primer *LOGICWARE*	50	45
MPROLOG P500 *by LOGICWARE*	495	395
MPROLOG P550 *by LOGICWARE*	220	175
Turbo PROLOG *by Borland Intl*	100	64
Turbo PROLOG Toolbox *by Borland Intl*	100	64

ai - smalltalk language

Smalltalk/V	99	84
EGA/VGA Color Option	50	45
Goodies Diskette	50	45
Smalltalk/Comm	50	45

ai - texas instruments

Arborist *Decision Tree Software*	*New*	595	519
PC Scheme *Lisp*		95	84
Personal Consultant Easy		495	435
Personal Consultant Image	*New*	495	435
Personal Consultant Online	*New*	995	869
Personal Consultant Plus		2950	2589
Personal Consultant Runtime		95	84

ada language

AdaVantage *by Meridian Software Systems*	*New*	795	735
AdaVantage Utility Packages	*New*	CALL	CALL
DOS Environment Package	*New*	50	47
Janus/ADA C Pak *by R&R Software*		95	84
Janus/ADA D Pak *by R&R Software*		1250	1059
Janus/ADA ED Pak *by R&R Software*		395	349

apl language

APL*PLUS/PC *by STSC*	595	424
APL*PLUS/PC Spreadsheet Mgr *by STSC*	195	139
APL*PLUS/PC Tools Vol 1 *by STSC*	295	199
APL*PLUS/PC Tools Vol 2 *by STSC*	85	58
ATLAS*GRAPHICS *by STSC*	450	329
Financial/Statistical Library *by STSC*	275	189
Pocket APL *by STSC*	95	69
STATGRAPHICS *by STSC*	795	579

assembly language

386 ASM/LINK *Cross Asm by Phar Lap*	495	389
8088 Assembler *w/Z-80 Translator by 2500 AD*	100	89
ASMLIB *Function Library by BC Assoc*	149	125
asmTREE *B-Tree Dev System by BC Assoc*	395	329
Cross Assemblers *Various by 2500 AD*	CALL	CALL
EZASM *by C Source* *New*	70	64
Microsoft Macro Assembler *New Version*	150	93
Norton Utilities *by Peter Norton*	100	CALL
Norton Utilities (Advanced)	150	99
Turbo Debugger *by Speedware* *New*	89	79
Turbo Editasm *by Speedware*	99	84
Visible Computer: 80286 *New*	100	89
Visible Computer: 8088 *by Software Masters*	80	64

basic language

87 Software Pak *by Hauppauge*	180	149
db/Lib *for QuickBASIC by AJS Publishing* *New*	99	89
EXIM Services Toolkit *by EXIM*	50	45
Finally *by Komputerwerks*	99	85
MACH 2 *by Micro Help*	69	55
Microsoft QuickBASIC *$20 Rebate Offer*	99	63
QBase *Relational Database by Crescent* *New*	89	79
Quick-Tools *by BC Associates*	130	109
QuickPak *by Crescent Software*	69	59
Scientific Subroutine Library *by Peerless*	125	99
Screen Sculptor *by Software Bottling*	125	91
Stay-Res *by MicroHelp*	69	55
True Basic *w/Run-time* *Special Price*	200	99
True Basic	100	79
Run-time Module	100	79
Various Utilities	50	41
Turbo BASIC Compiler *by Borland Intl*	100	64

blaise products

ASYNCH MANAGER *Specify C or Pascal*	175	135
C TOOLS PLUS	175	135
LIGHT TOOLS *for Datalight C* *Special Price*	100	65
PASCAL TOOLS	125	99
PASCAL TOOLS 2	100	79
PASCAL TOOLS & TOOLS 2	175	135
RUNOFF *Text Formatter*	50	45
TURBO ASYNCH PLUS	100	79
TURBO C TOOLS	129	CALL
TURBO POWER TOOLS PLUS	100	79
VIEW MANAGER *Specify C or Pascal*	275	199

borland products

EUREKA *Equation Solver*	167	105
Reflex & Reflex Workshop	200	128
Reflex *Data Base System*	150	89
Reflex Workshop	70	45
Sidekick & Traveling Sidekick	125	85
Sidekick	85	57
Traveling Sidekick	70	45
Superkey	100	64
Turbo BASIC Compiler	100	64
Turbo BASIC Database Toolbox *New*	100	64
Turbo BASIC Editor Toolbox *New*	100	64
Turbo BASIC Telecom Toolbox *New*	100	64
Turbo C Compiler *(Call for support products)*	100	64
Turbo Database Toolbox	70	41
Turbo Editor Toolbox	70	41
Turbo Gameworks Toolbox	70	41
Turbo Graphix Toolbox	70	41
Turbo Jumbo Pack	300	219
Turbo Lightning	100	64
Turbo PASCAL Numerical Methods Toolbox	100	64
Turbo PASCAL and Tutor	125	85
Turbo PASCAL	100	64
Turbo Tutor	40	24
Turbo PROLOG Compiler	100	64
Turbo PROLOG Toolbox	100	64
Word Wizard	70	47
Word Wizard and Turbo Lightning	150	94

c compilers

C86PLUS *by Computer Innovations*		497	375
DeSmet C *w/Debugger & Large case*		209	184
DeSmet C *w/Debugger only*		159	138
Eco-C *Complete System by Ecosoft*		140	119
Lattice C Compiler *vers. 3.2 from Lattice*		500	265
Mark Williams Let's C w/csd		125	99
Microsoft C Compiler w/CodeView	*New version*	450	269
Microsoft QuickC Compiler	*New*	99	63
Optimum-C *by Datalight*		139	105
Turbo C Compiler *by Borland*		100	64
Uniware 68000/10/20 Cross Compiler *by SDS*		995	829

c interpreters

C-terp *by Gimpel, Specify compiler*	*New Version*	300	219
C Trainer *with Book by Catalytix*		122	87
Instant C *by Rational Systems*	*New Version*	500	369
Introducing C *by Computer Innovations*		125	99
Run/C *by Age of Reason*		120	69
Run/C Professional *by Age of Reason*		250	145

c utilities

C++ *by Guidelines w/version 1.1 kernel*		195	172
Csharp Realtime Toolkit *by Systems Guild*	*New*	600	539
c-tree & r-tree Combo *by FairCom*		650	519
c-tree *ISAM File Manager*		395	315
r-tree *Report Generator*		295	239
Data Windows *by Magus Inc*	*New*	245	209
with Source Code	*New*	595	499
dBx *dBASE to C Translator by Desktop AI*		350	299
with Source Code	*New*	550	469
Flash-up Windows *by Software Bottling*		90	78
GraphiC *Color version by Sci Endeavors*		350	282
GRAFLIB *by Sutrasoft*		175	159
HALO *Graphics by Media Cybernetics*		300	205
HALO Development Pkg *for Microsoft*		595	389
The HAMMER *by OES Systems*		195	129
PANEL *Forms Management by Roundhill*		295	215
PANEL/TC *for Turbo C by Roundhill*	*New*	129	95
PANEL Plus *by Roundhill*		495	395
PC Lint *by Gimpel Software*		139	99
PLOTHP *by Sutrasoft*		175	159
RTC PLUS *Fortran to C by Cobalt Blue*		450	399
Scientific Subroutine Library *by Peerless*		175	135
TE Text Editor *source by Sub Systems*	*New*	95	85
Vitamin C *by Creative Programming*		225	158
VC Screen *Forms Designer*		100	79
Zview *by Data Mgmt Consultants*		245	139

cobol language

COBOLspll *by Flexus*		395	329
FPLIB *for Realia COBOL by BC Associates*		149	129
Micro Focus COBOL *See Micro Focus Section*			
Microsoft COBOL *See Microsoft Section*			
PCDT *by Pro-Code*	*New*	995	895
Realia COBOL with RealMENU	*New Version*	1145	899
Realia COBOL	*New Version*	995	783
RealCICS		995	783
RM/COBOL *by Ryan-McFarland*		950	CALL
RM/COBOL 85 *by Ryan-McFarland*		1250	CALL
SCREENIO *by Norcom*	*New*	400	379
screenplay *for COBOL by Flexus*		175	129

css products

Combo Package *by Custom Software Systems*	*New*	199	175
PC/SPELL *Spelling Checker*		49	45
PC/TOOLS *UNIX-like Utilities*		49	45
PC/VI *vi Editor*		149	99

debuggers & profilers

386 DEBUG *Cross Debugger by Phar Lap*		195	129
Advanced Trace-86 *by Morgan Computing*		175	115
Codesmith-86 *by Visual Age*		145	98
DSD87 *by Soft Advances*		125	79
MiniProbe *by Atron*		395	369
Periscope I *with Board by Periscope*		345	289
Periscope II *with NMI Breakout Switch*		175	139
Periscope II-X *Software only*		145	105
Periscope III *8 MHz version*		995	799
Periscope III *10 MHz version*		1095	899
The PROFILER *with Source Code by DWB*		125	89
TURBOsmith *Source debugger for Turbo Pascal*		69	59
The WATCHER *Profiler by Stony Brook*		60	51

disk utilities

Back-It *by Gazelle Systems*		100	89
Disk Optimizer *by Softlogic Systems*		60	55
FASTBACK *by 5th Generation Systems*		179	129
Vcache *by Golden Bow Systems*	*New*	50	47
Vopt *by Golden Bow Systems*	*New*	50	47
Vfeature *by Golden Bow Systems*	*New*	80	74
Vfeature Deluxe *by Golden Bow Systems*	*New*	120	111
XenoCopy-PC *by XenoSoft*		80	69

dos utilities

Command Plus *by ESP Software*		80	69
FANSI-CONSOLE *by Hersey Micro*		75	62
MicroHelp Utilities *by MicroHelp*		59	49
Norton Commander *by Peter Norton*		75	CALL
OPAL *Shell Language by Software Factory*		99	89
Q-DOS II *by Gazelle Systems*		70	59
Taskview *by Sunny Hill Software*		80	55

essential products

C Utility Library		185	119
Essential Comm Library *with Debugger*		250	189
Essential Comm Library *Software Only*		185	125
Breakout Debugger Only *Any language*		125	89
Essential Graphics		250	183

forth language

CFORTH *Native Code Compiler by LMI*		300	229
Forth/83 Metacompiler *Specify Target*		750	599
PC/Forth *by Laboratory Microsystems*		150	109
PC/Forth+ *by Laboratory Microsystems*		250	199
Advanced Color Graphics Support		100	74
Enhanced Graphics Support		200	148
Intel 8087 Support		100	74
Interactive Symbolic Debugger		100	74
Native Code Optimizer		100	74
Software Floating Point		100	74
UR/Forth *by LMI*		350	279
Object Module Libraries		500	395

fortran language

50 MORE: FORTRAN *by Peerless Scientific*		125	95
ACS Time Series *by Alpha Computer Service*		495	389
AUTOMATED PROGRAMMER *by KGK Automated*	*New*	CALL	CALL
Essential Graphics *by Essential Software*		250	183
For-Winds *by Alpha Computer Service*		90	69
Forlib-Plus *by Alpha Computer Service*		70	44
FORTLIB *by Sutrasoft*		125	109
FORTRAN Addendum *by Impulse Engr*		95	85
FORTRAN Addenda *by Impulse Engr*		165	138
GRAFLIB *by Sutrasoft*		175	159
HALO *Graphics by Media Cybernetics*		300	205
I/O PRO w/No Limit Library *by MEF*		250	219
Microcompatibles Combo Package		240	215
Grafmatic		135	117
Plotmatic		135	117
Microsoft FORTRAN *w/CodeView*		450	269
No Limit Library *by MEF Environmental*		129	109
Numerical Analyst *by MAGUS*		295	249
PANEL *by Roundhill Computer Systems*		295	215
PLOTHP *by Sutrasoft*		175	159
RM/FORTRAN *by Ryan-McFarland*		595	CALL
RTC PLUS *Fortran to C by Cobalt Blue*		450	399
Scientific Subroutine Lib *by Peerless*		175	135
Statistician *by Alpha Computer Service*		295	235
Statlib.GL: *by Peerless*	*New Version*	295	239
Statlib.TSF: *by Peerless*	*New Version*	295	239
Strings & Things *by Alpha Computer Service*		70	45

greenleaf products

Greenleaf Comm Library		185	125
Greenleaf Data Windows Library		225	155
with Source Code		395	249
Greenleaf Functions		185	125

help utilities

HELP/Control *by MDS*		125	99
On-line Help *from Opt-Tech*		149	99
SoftScreen/HELP *by Dialectic Systems*		195	149

lattice products

Lattice C Compiler *ver 3.2 from Lattice*		500	265
with Library Source Code		900	495
C Cross Reference Generator		50	37
with Source Code		200	139
C-Food Smorgasbord *Function Library*		150	95
with Source Code		300	179
C-Sprite *Source Level Debugger*		175	119
Curses *Screen Manager*		125	85
with Source Code		250	169
dBC *Specify dBC II or dBC III*		250	169
with Source Code		500	356
dBC III Plus		750	594
with Source Code		1500	1184
LMK *Make Facility*		195	138
RPG II Combo *All three items below*		1100	875
RPG II Compiler *No Royalties*		750	625
RPG II SEU *Screen Entry Utility*		250	199
Sort/Merge		250	199
RPG II Screen Design Aid Utility		350	309
SecretDisk *File Encryption Utility*		120	88
SideTalk *Resident Communications*		120	88
SSP/PC *Scientific Subroutine Library*		350	269
Text Management Utilities		120	88
TopView Toolbasket *Function Library*		250	178
with Source Code		500	356

metagraphics products

LightWINDOW/C *for Datalight C*		95	79
FontWINDOW		95	79
FontWINDOW/PLUS		275	229
MetaWINDOW *No Royalties*		195	159
MetaWINDOW/PLUS		275	229
TurboWINDOW/C *for Turbo C*		95	79
TurboWINDOW/Pascal *for Turbo Pascal*		95	79

micro focus products

Micro Focus Level II COBOL *w/Animator*		495	395
Level II COBOL		349	279
Level II Animator		195	155
Micro Focus Level II COBOL/ET for UNIX		CALL	CALL
Micro Focus Personal COBOL		149	119
Micro Focus Professional COBOL		2000	1595
Micro Focus VS COBOL/XENIX		1495	1195

Micro Focus Support Products:			
COBOL/IQ *Ad hoc Report Writer*		495	395
COBOL/IQ *for DOS 3.X Networks*		995	795
FORMS-2		295	235
SOURCEWRITER		995	795

microport products

System V/386 Combination	*New*	799	699
386 Runtime System	*New*	199	169
386 Software Development System	*New*	499	429
Text Preparation System	*New*	199	169
386 Unlimited License Kit	*New*	249	209
DOSMerge386 *Run DOS and UNIX together*	*New*	CALL	CALL
System V/AT Combination		549	465
AT Runtime System		199	169
AT Software Development System		249	209
Text Preparation System		199	169
AT Unlimited License Kit		249	209
DOSMerge286 *Run DOS and UNIX together*	*New*	149	129

microsoft products

Microsoft BASIC Compiler *for XENIX*		695	419
Microsoft BASIC Interpreter *for XENIX*		350	209
Microsoft C Compiler *w/CodeView*	*New Version*	450	269
Microsoft COBOL Compiler *with COBOL Tools*		700	429
for XENIX		995	609
Microsoft FORTRAN *Optimizing Compiler/CodeView*		450	269
Microsoft FORTRAN *for XENIX*		695	419
Microsoft Learning DOS		50	36
Microsoft LISP *Common LISP*		250	149
Microsoft MACH 10 *with Mouse & Windows*		549	369
Microsoft MACH 10 *Board only*		399	279
Microsoft Macro Assembler	*New Version*	150	93
Microsoft Mouse *for IBM PS/2*	*New*	175	114
Microsoft Mouse *Bus Version*		175	114
Microsoft Mouse *Serial Version*		195	124
Microsoft muMath *Includes muSIMP*		300	179
Microsoft Pascal Compiler		300	179
for XENIX		695	419
Microsoft QuickBASIC	*$20 Rebate Offer*	99	63
Microsoft QuickC	*New*	99	63
Microsoft Sort		195	125
Microsoft Windows		99	63
Microsoft Windows Development Kit		500	299
Microsoft Word		450	269

modula-2 language

EXE2LNK *MASM Interface by PMI*	*New*	49	45
Macro2 *Macro preprocessor by PMI*	*New*	89	79
ModBase *by PMI*	*New*	89	79
MODULA-2 Apprentice Pkg by LOGITECH		99	79
MODULA-2 Magic Pkg by LOGITECH		99	79
MODULA-2 ROM Pkg & Cross RT Debugger		299	239
MODULA-2 Window Pkg by LOGITECH		49	39
MODULA-2 Wizard's Pkg by LOGITECH		199	159
Repertoire *by PMI*		89	75

mouse products

LOGIMOUSE BUS *with PLUS Pkg by LOGITECH*		119	98
with PLUS & PC Paintbrush		149	119
with PLUS & CAD Software		189	153
with PLUS & CAD & Paint		219	179
LOGIMOUSE C7 *with PLUS Pkg, Specify Connector*		119	98
with PLUS & PC Paintbrush		149	119
with PLUS & CAD Software		189	153
with PLUS & CAD & Paint		219	179
Microsoft Mouse *See Microsoft Section*			

other languages

ACTOR *by Whitewater Group*	*New*	495	419
CCS MUMPS *Single-User by MGlobal*		60	50
CCS MUMPS *Single-User Multi-Tasking*		150	129
CCS MUMPS *Multi-User*		450	359
Marshal Pascal *by Marshal Language Systems*		189	165
Pascal-2 *by Oregon Software*		395	325
Personal REXX *by Mansfield Software*		125	99
SNOBOL4+ *by Catspaw*		95	80

other products

Carbon Copy *by Meridian Technology*	*New*	195	179
Dan Bricklin's Demo Pgm *by Software Garden*		75	57
Dan Bricklin's Demo Tutorial		50	45
Fast Forward *by Mark Williams*	*New*	70	59
First Publisher *with Mouse from Logitech*	*New*	CALL	CALL
Informix *All Varieties by Informix*		CALL	CALL
Instant Replay *by Nostradamus*		150	CALL
Mace Utilities *Paul Mace Software*	*New*	99	89
MKS Toolkit *w/vi Editor by MKS*	*New Version*	139	114
MicroTEX *Typesetting from Addison Wesley*		295	CALL
Net-Tools *by BC Associates*		149	129
OPT-Tech Sort *by Opt-Tech Data Proc*		149	99
PC/TOOLS *by Custom Software*		49	45
Screen Machine *by MicroHelp*		79	59

phoenix products

Pasm86 *Macro Assembler version 2.0*		195	108
Pdisk *Hard Disk & Backup Utility*		145	99
Pfantasy Pac *Phoenix Combo*		995	595
Pfinish *Execution Profiler*		395	209
Pfix86plus *Symbolic Debugger*		395	209
PforCe *Specify C Compiler*		395	209
PforCe++ *Specify C Compiler and C++*		395	209
Plink86plus *Overlay Linker*		495	275
Pmaker *Make Utility*		125	78
Pmate *Macro Text Editor*		195	108
Pre-C *Lint Utility*		295	154
Ptel *Binary File Transfer Program*		195	108

polytron products

PolyBoost *The Software Accelerator*		80	64
PolyDesk III		99	72
PolyDesk III Archivist		50	42
PolyDesk III Cryptographer		50	42
PolyDesk III Talk		70	52
PolyLibrarian *Library Manager*		99	89
PolyLibrarian II *Library Manager*		149	129
PolyMake *UNIX-like Make Facility*		149	129
PolyShell		149	129
Polytron C Beautifier		50	45
PolyXREF *Complete Cross Ref Utility*		219	185
PolyXREF *One language only*		129	109
PVCS Corporate *Version Control System*		395	329
PVCS Personal		149	129

program mgmt utilities

Interactive EASYFLOW *by Haventree*		150	125
PrintQ *by Software Directions*		89	84
Quilt Computing Combo Package		250	199
QMake *Program Rebuild Utility*		99	79
SRMS *Software Revision Mgmt System*		185	159
Source Print *by Aldebaran Labs*		97	75
TLIB *Version Control System by Burton*		100	89
Tree Diagrammer *by Aldebaran Labs*		77	67

raima products

dbQUERY *Single-User Query Utility*		195	129
Single-User with Source Code		495	389
Multi-User		495	389
Multi-User with Source Code		990	699
dbVISTA *Single-User DBMS*		195	129
Single-User with Source Code		495	389
Multi-User		495	389
Multi-User with Source Code		990	699

sco products

Complete XENIX System V *by SCO*		1295	994
Development System		595	499
Operating System *Specify XT or AT*		595	499
Text Processing Package		195	144
Lyrix *by SCO*		595	449
SCO Professional *1-2-3 Workalike for XENIX*		795	595
SCO XENIX-NET		595	495

softcraft products

Btrieve *ISAM Mgr with No Royalties*		245	184
Xtrieve *Query Utility*		245	184
Report Option *for Xtrieve*		145	99
Btrieve/N *for Networks*		595	454
Xtrieve/N		595	454
Report Option/N *for Xtrieve/N*		345	269

text editors

Brief & dBrief Combo *from Solution Systems*		275	CALL
Brief		195	CALL
dBrief *Customizes Brief for dBASE III*		95	CALL
Epsilon *Emacs-like editor by Lugaru*		195	147
KEDIT *by Mansfield Software*		125	98
Micro/SPF *by Phaser Systems*		175	139
Microsoft Word		450	269
PC/VI *by Custom Software Systems*		149	105
SPF/PC *by Command Technology Corp*		CALL	CALL
Vedit Plus *by CompuView*		185	128

turbo pascal utilities

ALICE *Interpreter by Software Channels*		95	66
DOS/BIOS & Mouse Tools *by Quinn-Curtis*		75	67
Flash-up Windows *by Software Bottling*		90	78
MACH 2 *for Turbo Pascal by Micro Help*		69	55
MetraByte D/A Tools *by Quinn-Curtis*		100	89
Science & Engrg Tools *by Quinn-Curtis*		75	67
Screen Sculptor *by Software Bottling*		125	91
Speed Screen *by Software Bottling*		35	32
System Builder *by Royal American*		150	129
IMPEX *Query Utility*		100	89
Report Builder		130	115
TDebugPLUS *by TurboPower Software*		60	49
Tmark *by Tangent Systems*	*New*	80	69
Turbo EXTENDER *by TurboPower Software*		85	64
Turbo OPTIMIZER *by TurboPower*		75	65
Turbo OPTIMIZER *with Source Code*		125	108
Turbo Professional *by Sunny Hill*		70	45
Turbo.ASM *by BC Associates*	*New*	100	89
TurboHALO *from IMSI*		129	98
TurboPower Utilities *by TurboPower*		95	78
TurboRef *by Gracon Services*		50	45
TURBOsmith *Source Debugger by Visual Age*		69	59
Universal Graphics Library *by Quinn-Curtis*		150	119

wendin products

Operating System Toolbox		99	79
PCNX *Operating system*		99	79
PCVMS *Similar to VAX/VMS*		99	79
Wendin-DOS *Multitasking DOS*	*New*	99	85
XTC *Text Editor w/Pascal source*		99	75

xenix/unix products

Btrieve *ISAM File Mgr by SoftCraft*		595	454
C-terp *by Gimpel, Specify compiler*		498	379
c-tree *ISAM Mgr by FairCom*		395	329
dBx *with Library Source by Desktop AI*		550	469
Desqview *from Quarterdeck*	*New*	100	85
DOSIX *Console Version by Data Basics*		399	349
DOSIX *User Version by Data Basics*		199	179
Informix *All Varieties by Informix*		CALL	CALL
Micro Focus Products *See Micro Focus Section*			
Microport Products *See Microport Section*			
Microsoft Products *See Microsoft Section*			
PANEL Plus *by Roundhill Computer Systems*		795	535
REAL-TOOLS *Binary Version by PCT*		149	89
Library Source Version		399	289
Complete Source Version		999	729
RM/COBOL *by Ryan-McFarland*		1250	CALL
RM/FORTRAN *by Ryan-McFarland*		750	CALL
SCO Products *See SCO Section*			

Call or write for our FREE comprehensive price guide.

LOWEST PRICES

Due to printing lead times, some of our current prices may differ from those shown here. Call for latest pricing.

FREE SHIPPING

Orders within the USA (including Alaska & Hawaii) are shipped FREE via UPS. Express shipping is available at the shipping carrier's standard rate with no rush fees or handling charges. To avoid delays when ordering by mail, please call first to determine the exact cost of express shipping.

CREDIT CARDS

VISA and MasterCard are accepted at no extra cost. Your card is charged when your order is shipped. Mail orders please include credit card expiration date and authorized signature.

CODs AND POs

CODs and Purchase Orders are accepted at no extra cost. No personal checks are accepted on COD orders. POs with net 30-day terms (with initial minimum order of $100) are available to qualified US accounts only.

SALES TAX

Orders outside of Ohio are not charged state sales tax. Ohio customers please add 6% Ohio tax or provide proof of tax-exemption.

INTERNATIONAL ORDERS

Shipping charges for International and Canadian orders are based on the shipping carrier's standard rate. Since rates vary between carriers, please call or write for the exact cost. International orders (except Canada), please include an additional $10 for export preparation. All payments must be made with US funds drawn on a US bank. Please include your telephone number when ordering
Due to government regulations, we cannot ship to all countries.

VOLUME ORDERS

Volume orders may qualify for additional discounts. Call us for special pricing.

SOUND ADVICE

Our knowledgeable technical staff can answer technical questions, assist in comparing products and send you detailed product information tailored to your needs.

30-DAY GUARANTEE

Most of our products (excluding books) come with a 30-day documentation evaluation period or a 30-day return guarantee. Please note that some manufacturers restrict us from offering guarantees on their products. Call for more information.

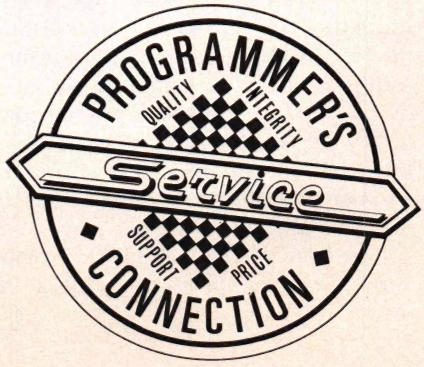
MAIL ORDERS

Please include your telephone number on all mail orders. Be sure to specify computer, operating system and any applicable compiler or hardware interface(s). Send mail orders to:

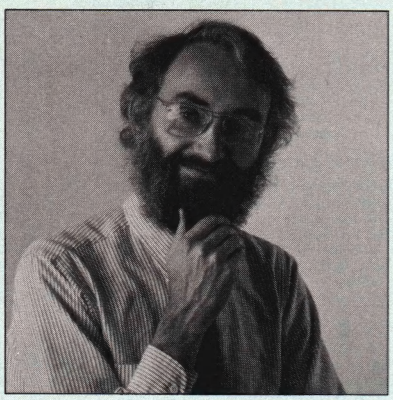
Programmer's Connection
Order Processing Department
136 Sunnyside Street
Hartville, OH 44632

USA	**800-336-1166**
CANADA	**800-225-1166**
OHIO & ALASKA (Collect)	**216-877-3781**
TELEX	**9102406879**
EASYLINK	**62806530**
INTERNATIONAL	**216-877-3781**
CUSTOMER SERVICE	**216-877-1110**

Hours: Weekdays 8:30 AM to 8:00 PM EST.



SWAINE'S FLAMES



It seems that look and feel may be less of a problem for Lotus Development Corp. than compiled spreadsheets are. Several products now turn 1-2-3 spreadsheet data into executable programs, providing a legal way to share and manipulate spreadsheet data while only buying one copy of the software. What you can't do with these programs is design, or alter the design of, spreadsheets. In a company where spreadsheets are designed by one person and used by many, these products could save a lot of bucks—bucks that Lotus may feel ought to be flowing its way. Should be interesting.

Personally, I find it weird to think of 1-2-3 as a programming environment.

Speaking of weird programming environments, terminate-and-stay-resident implementations of BASIC and other programming languages are popping up all over the place. If you ask me, anyone who develops software in TSR space is playing with fire, but that won't stop the crazed code junkies from snorting this stuff up.

Well, you might program in the ether, but you don't mess around with the herald of Galactus. Marvel Comics has leaned on Acius about the name Silver Surfer, and the company's Mac database is now called 4th Dimension. In an unrelated development in the same paragraph, Living Videotext recently moved to Easy Street (really), and there were rumors that the company was about to be bought by Symantec. More interesting were the hints that the vendor of a very successful windowing system was being pressured by a well-known venture capitalist to sell out to a major operating system vendor for the good of the industry.

And you thought the Ollie North Show was the best soap opera in town.

Here, for all you model programmers, is what you need to do to become a gatefold in a major computer magazine: As a result of your strategic partnership with the largest computer company, you deliver a multitasking operating system (retaining the right to license it to your partner's competitors), and when your partner lays out its plans to add competitive value to the product with connectivity extensions, you strike another strategic partnership with a leading LAN vendor, intended to give you a piece of everybody else's answer to those extensions. Meanwhile you hedge your bets on the operating system itself by striking a strategic partnership with a leading multiuser operating system vendor, and when you've got that all sewn up, you convince programmers to pay you thousands of dollars to learn how to develop the software to run under your operating system.

Cousin Corbett's Secrets of Software Success, Part II: Product Names.

The following advice from my cousin Corbett's forthcoming book is reprinted here without comment.

"Although there is no formula for successful product naming, there are a few simple rules that will help you avoid the most common mistakes.

"Try for unambiguous pronounceability. Unless you're selling sexually explicit software, people ought to be able to go into a store and ask for your product without embarrassment. Does Digitalk expect people to ask for Smalltalk-vee or Smalltalk-five? (Some programmers have taken to calling it ess-tee-vee.) TEX shouldn't be spelled like tecks if it's to be pronounced teck. Vision is pronounced vih-zhun, not vih-zee-on, and any attempt to get masses of people to pronounce it wrong will meet with resistance.

"By all means use your own name, especially if it also happens to have other, favorable connotations, as Norton suggests the authoritative *Norton Anthologies*. Consider changing your name to Webster or Roget. Or Knuth."

And this is from Ray Duncan:

_i_hael s_aine
editor
dr. do__.s _ournal
_i_e _ero one _al_eston dr.
red_ood _it_. _a
nine _our_ero si_ three
dear _i_hael.
_our .s_aine.s _ la_es. _olu_n in
the _ul_ one nine ei_ht se_en dd_
_as _uite thou_ht.
_ro_o_in_. the _on_e_t o_ an
en_r__tion _ethod that _an.t
_e used su__ess_ull_ _ithout
_oth a _a_hine .to en_r__t. and a
hu_an .to de_r__t. _as
uite ne to _e. it o__urs to _e
that _erha_s this rather si__le
t__e o_ en_r__tion _an
_e_o_e the _asis o_ a _o_er_ul
et eas to e__lain .to la __en. test
or .understandin. o_ natural
lan_ua_e __ _a_hines.
sin _erel_ _ours.
ra _ dun_an

Michael Swaine
editor-in-chief

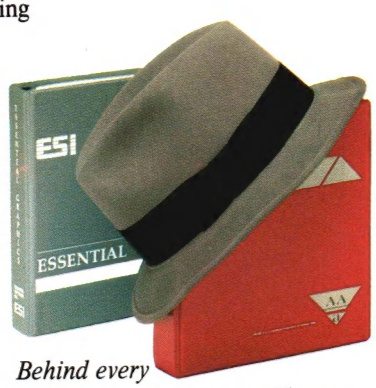
Some Very Impressive People Keep Our Asynch C Tools Under Their Hats

This is the only way we can get some of our customers to take their hats off to us in public. We understand. Most people like to keep a good thing to themselves.

Once you see how powerful and simple our functions are, you'll want to treat our Communications Library Plus as a well guarded trade secret too.

Essential provides the best alternative to Assembly language for communicating via an RS-232.

However, should curiosity get the best of you, call us at 201-762-6965 and we'll let you in on the identity of some of our secret admirers.



Behind every great program is a great library

What Good Are Library Functions If You Can't Get Them To Work?

Providing functions is not enough. Essential Communications Library Plus includes BreakOut, a slick on-line data monitor. BreakOut saves hours of frustration. We know, we used it to debug the XMODEM and other functions in Essential Communications.

No Royalties, 30-Day Guarantee

If within 30 days you don't find our library or BreakOut totally satisfactory, hang the whole thing up and receive a complete refund.

Functions At A Glance

- Interrupt driven to 9600 baud
- Hayes compatible support
- 150 page manual—tutorial
- XMODEM, XON/XOFF support
- Timer/Keyboard functions
- Input buffers to 500K
- Source included
- Demo terminal program
- Demo BBS system
- All major compilers supported
- Ctrl-Break status

Comm Library Plus/BreakOut $250 **Comm Library $185**

BreakOut On-line Monitor/Debugger

- Monitor RS232 ports up to 9600 baud
- Control comm variables and line signals
- Send/receive data using the scratch-pad editor
- Edit scratch-pad in Hex or ASCII
- Save data to file or re-transmit it
- User configurable keyboard macros
- Use symbols for control chars (<ACK>) for Hex 06

BreakOut $125

Do Your Homework

The library you buy will influence the rest of your programming life. When you've done your homework, you'll choose Essential. Call our support staff of C programmers and find out now how things will be after your check clears.

To order or for support call: 201-762-6965

For foreign orders contact:

England: Gray Matter Tel. (0364) 53499
Japan: Lifeboat Inc. of Japan Tel: 293 4711
West Germany: Omnitex Tel. 07623-61820

Essential Software, Inc.

P.O. Box 1003, Maplewood, New Jersey 07040

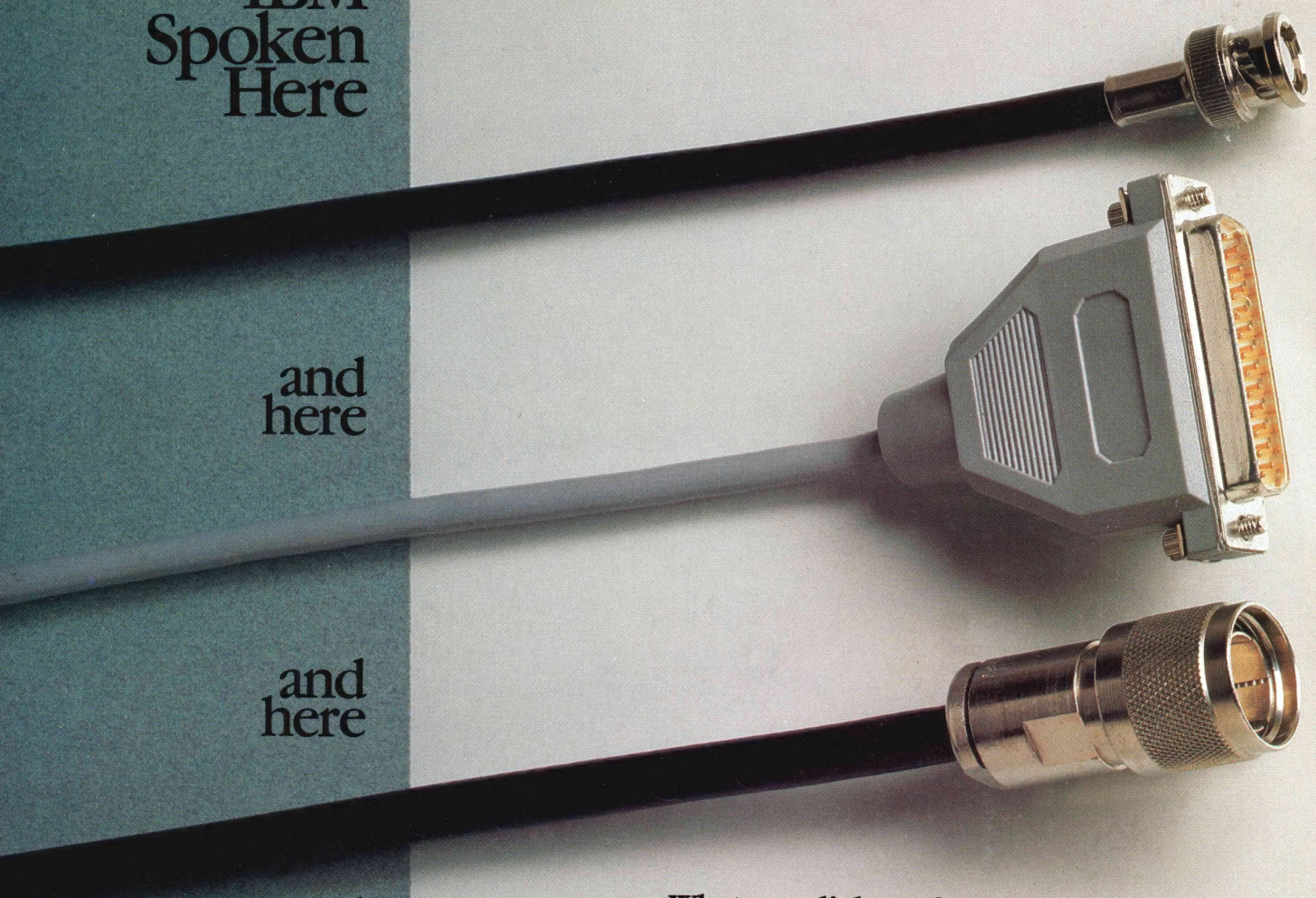
CIRCLE 139 ON READER SERVICE CARD

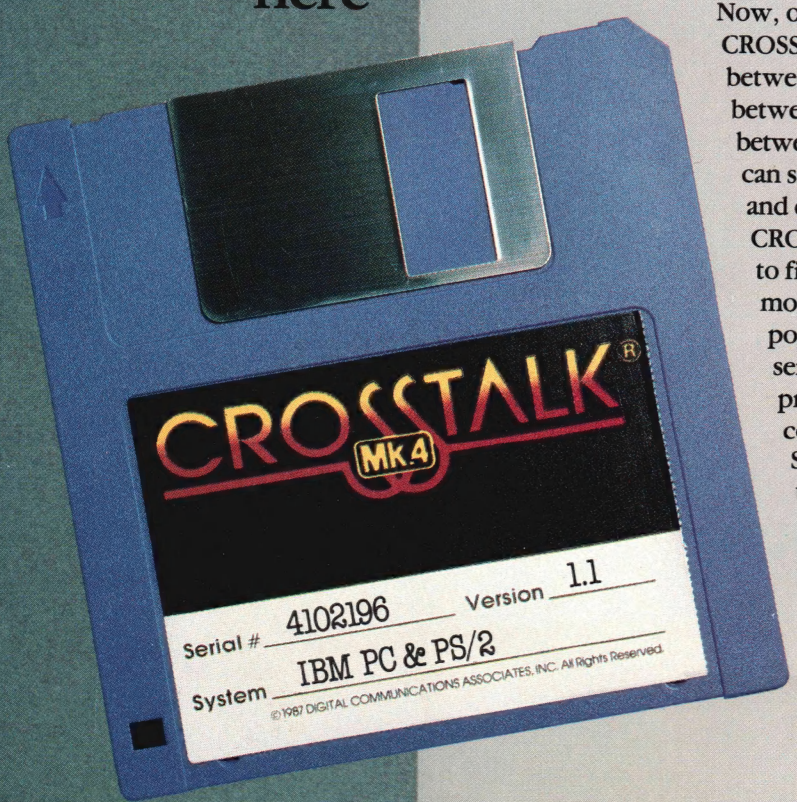
IBM Spoken Here

and here

and here

and here





Whatever dialect of IBM you need to speak, CROSSTALK® Mk. 4 makes the connection.

Now, one program does the job that used to require several. CROSSTALK® Mk. 4 allows high-speed direct communications between PCs and minicomputers, or (with an IRMA™ board) between your PC and an IBM Mainframe, or (with Smart Alec between your PC and IBM System 3x's. If you like, CROSSTAL can support all of these sessions (and others) simultaneously and display each session in its own window.

CROSSTALK Mk. 4 emulates all the terminals you're likely to find useful. That includes IBM 3101 (page and character modes), IBM 525x, IBM 529x, IBM 327x, as well as many popular async terminals like the DEC VT100 and VT220 series. CROSSTALK Mk. 4 includes the powerful CASL™ programming language, which allows you to automate communications applications quickly and easily.

So if you're used to thinking of CROSSTALK just to use with a modem, you're missing some important connections. Ask your dealer for details, or write:

dca® Digital Communications Associates, Inc.
1000 Holcomb Woods Parkway / Roswell, Georgia 30076
1-800-241-6393

CROSSTALK® COMMUNICATIONS

CIRCLE 171 ON READER SERVICE CARD